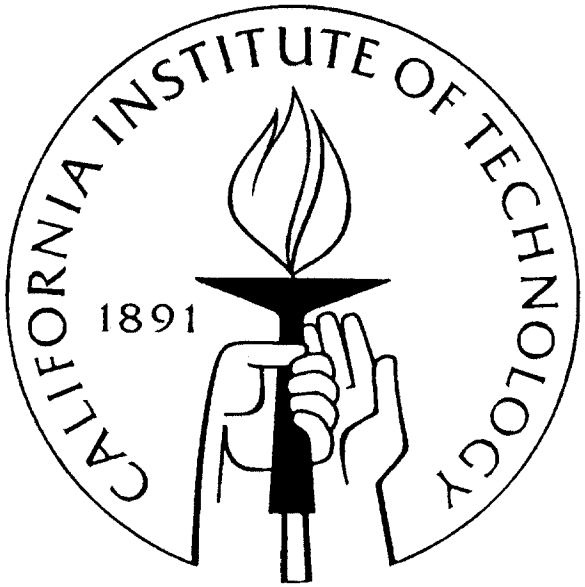


Communication Behavior of Linear Arrays of Processes

Tak K Lee

Computer Science Department
California Institute of Technology

Caltech-CS-TR-89-13

Communication Behavior of

Linear Arrays of Processes

by
Tony Lee

June 8, 1988

Abstract

This paper investigates the *communication behavior* of a linear array of processes, each process implementing the same *program*. For programs with *cyclic* communication patterns, simple criteria for determining whether they induce *constant response time* on the array are established. Also, an algorithm is developed for characterizing programs with more general communication patterns.

Contents

List of Figures

Chapter 1

Introduction

1.1 Communicating Sequential Processes

It has long been recognized that many computations can be performed efficiently by using networks that consist of a large number of processes with all processes basically performing the same operations [2, 3]. In this paper, we shall study one such type of networks, namely, linear arrays of processes. Our main goal is to determine how the communication behavior among the individual processes affects the overall performance of a given network.

We shall start by giving an example on how our model can be used to solve a simple problem.

For an integer N and a sequence of numbers $a[0]$, $a[1]$, ..., the i^{th} running sum of size N is defined by

$$b[i] = \sum_{j=i-N+1}^{i} a[j]$$

where $a[j] = 0$ if $j < 0$.

The sequence of running sums can be computed by the concurrent execution of a set of N processes arranged in a linear array; Figure 1.1 illustrates the process connections when $N = 3$. All of the processes – $p[0], p[1], \ldots, p[N-1]$ – perform the same sequence of operations which will be specified later. A process can interact with another process only by executing **communication actions** on a **channel** connecting the two processes. As indicated in the illustration, each process $p[l]$ communicates only with the neighbor immediately on its left and the neighbor immediately on its right. The environment, also denoted as $p[-1]$ or $p[N]$, is considered as the left neighbor of $p[0]$ and the right neighbor of $p[N-1]$.

In our model, each channel connects exactly two processes and is labelled by a pair of **ports** – one port associated with each process. In this example, the channels are labelled as $\langle p[i-1].ar, p[i].al \rangle$ and $\langle p[i-1].br, p[i].bl \rangle$, for $i = 0, \ldots, N$.

Next, we choose to describe the operations performed by each process by the following **process description** P:

$$P \equiv xold := 0\ ;\ *[al?x\ ;\ ar!xold\ ;\ br?y\ ;\ bl!(x+y)\ ;\ xold := x].$$

The notations used are adopted from [5]. The semicolon is the sequencing operator between statements, the symbol ":=" is the assignment operator, and the "*[...]" construct is used

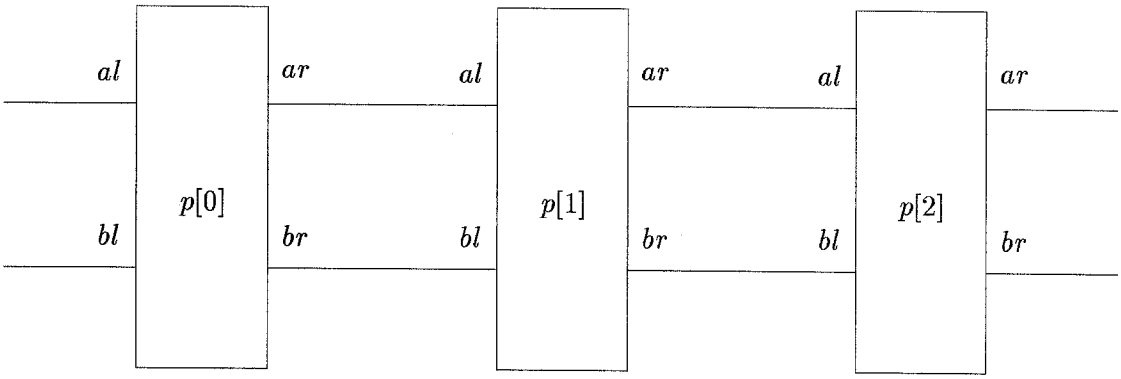


Figure 1.1: Process Connections for Computing Running Sums

to denote endless repetition of the enclosed statements. The statement "$al?x$" denotes the communication action for the input of a value from port "al" of the process into the variable "x". Similarly, the statement "$bl!(x + y)$" represents the output of the value of the expression "$(x + y)$" to port "bl" of the process.

The main focus of this paper will be on zero-slack channels [4]. In other words, for a given channel, the number of communication actions completed on a given port is exactly the same as the number of communication actions completed on the corresponding port. If one process attempts to perform a communication action before its neighbor performs the corresponding communication action, then the former process will be **suspended** until the latter is ready. In the final section of this paper, we shall give some results for channels with *one-sided infinite slacks*, *i.e.*, channels satisfying the property that the number of communcation actions completed on the input port is no greater than the number of communication actions completed on the output port.

Returning to the running sum example, for $i \geq 0$, let $a[i]$ be the i^{th} value input by $p[0]$ via "$al?x$", and $b[i]$ be the i^{th} value output by $p[0]$ via "$bl!(x + y)$". If we assume that the environment outputs only zeros on its "bl" port (*i.e.*, "$br?y$" in $p[N-1]$ is equivalent to "$y := 0$"), then it is not too difficult to see that for $p[0] \,||\, p[1] \,||\, p[2]$, the parallel execution of the N processes, $b[i]$ is the i^{th} running sum of size N for the sequence $a[0]$, $a[1]$, ...

Note, however, that for $N = 3$, the following sequence of operations is involved in the computation:

1. $p[0]$ receives a value of $a[i]$ and sends $a[i-1]$ to $p[1]$; $p[0]$ is then suspended on the $br?y$ action.

2. $p[1]$ sends $a[i-2]$ to $p[2]$ and is then suspended.

3. $p[2]$ sends $a[i-2] + 0$ back to $p[1]$.

4. $p[1]$ sends $a[i-1] + a[i-2]$ back to $p[0]$.

5. $p[0]$ finally outputs $b[i] = a[i] + a[i-1] + a[i-2]$.

Thus, at least four communications must be performed between the input of $a[i]$ and the output of $b[i]$; it should be clear that this number increases as N increases.

Next, suppose that, instead of P, the operations performed by each of the processes in Figure 1.1 is specified by a new process description P':

$$P' \equiv y := 0; *[al?x; bl!(x + y); ar!x; br?y].$$

Assuming the process connections, environmental interface, and naming conventions remain the same, then, though this time with slightly more difficulty, it can be shown that $b[i]$ is still the i^{th} running sum of size N for the sequence $a[0]$, $a[1]$, ...

However, the sequence of operations involved in the computation is different from the previous case. After $p[0]$ has received $a[i]$, $b[i]$ is output immediately and the calculation for $b[i+1]$ is initiated before $a[i+1]$ is input. In fact, it can be shown that P' induces pipelining and that partial results are computed in anticipation of their eventual usage, thus leading to **constant response time** between the arrival of the input and the production of the successive output, regardless of the number of processes involved. Moreover, the utilization of the resources is much higher for this second example in that all three processes may be actively processing at the same time; whereas, in the previous example, $p[0]$ basically waits idly while values propagate to $p[N-1]$ and back.

Although these examples are fairly artificial, they do serve to illustrate how the ordering of communication actions performed by each process can have a significant effect on the throughput and resource utilization for the entire network. The main purpose of this paper is to develope a complete characterization of communication patterns that induce constant response time for linear arrays of concurrent processes.

1.2 Outline of Paper

In Chapter 2, formal notations for specifying programs and general definitions are introduced — most of these concepts, especially that of **sequence function**, are adapted from [6]. Chapter 3 analyzes processes with **cyclic** communication behavior, and Chapter 4 investigates the response time of programs with **pseudo-cyclic** communication behavior. Chapter 5 extends the results to arrays under the infinite-slack communication model. Finally, Chapter 6 serves as a summary and conclusion.

Note that the algebraic specifications for the algorithms described in this paper are relegated to the appendix. Instead, a more "intuitive" paraphrase of each algorithm and an outline for proving its correctness are included in the main text. Also, graphical representations are used as an aid in presenting the results; the proofs themselves are algebraic and do not rely on these references.

Chapter 2

Notations and Definitions

2.1 Programs

A linear array is modelled by an integer N, the number of processes, and a **program** $\mathcal{P}$ that specifies both the channel connections and the communication behavior of the generic process. To formalize the communication behavior, trace theoretical notations and concepts are adopted from [6]. First, we restrict ourselves to only deterministic, **data-independent** processes; *i.e.*, the communication behavior of each process does not depend on the values of the data transmitted. Therefore, internal computations and values being input or output have no observable effect on the sequencing of communication actions between processes and can be ignored in the analysis. Thus, for each program, we consider only its communication actions, which will be represented by **symbols** α_0, α_1, etc.

Since we are dealing with data-independent processes, the communication behavior of a program can be specified by a single (possibly non-terminating) sequence of communication actions. The sequence of symbols corresponding to this sequence will be denoted by Γ. Also, at a given time, the sequence of communication actions performed by a process thus far is finite and can therefore be represented by a **trace**, which is a finite string of symbols. Furthermore, the set of all possible sequences of communication actions performed by a process is precisely **pref**(Γ), the set of all prefix strings of Γ.

Finally, we will need to model the interaction between adjacent processes. This is done via the notion of a **correspondence**, which is an ordered pair of symbols. Intuitively, the correspondence (α_e, α_f) implies that the i^{th} occurrence of α_e in process $p[l]$ "coincides" with the i^{th} occurrence of α_f in process $p[l+1]$. The precise nature of a correspondence will be clarified later.

The considerations above lead to the following definition:

Definition 2.1 *A* **program** $\mathcal{P}$ *is a triple* $\langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ *where*

- $\mathrm{A} = \{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$ *is called the* **alphabet** *of* $\mathcal{P}$*, and each* α_i *is a* **symbol** *in the alphabet;*
- Γ *is a (possibly non-terminating) sequence of symbols of* A. Γ *is called the* **communication pattern** *of* $\mathcal{P}$*;*
- $\mathrm{C} \subseteq \mathrm{A} \times \mathrm{A}$ *is the* **set of correspondences** *for* $\mathcal{P}$.

Example 2.1: Consider the representation of the program P described in the previous section. First, the alphabet will be $\mathrm{A} = \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}$ with the symbol α_0 denoting the "*al*?" communication action, α_1 denoting "*ar*!", α_2 denoting "*br*?", and α_3 denoting "*bl*!". Note that internal computations and values communicated are ignored.

Next, the communication pattern of $\mathcal{P}$ is represented by

$$\Gamma = (\alpha_0\alpha_1\alpha_2\alpha_3)^\star,$$

with the star denoting endless repetition. Finally, to describe the fact that $p[l]$ completes its "*ar*!" action "at the same time" that $p[l+1]$ completes its "*al*?" action, we specify that $(\alpha_1, \alpha_0) \in \mathrm{C}$. Similarly, "*br*?" corresponds to "*bl*!"; so,

$$\mathrm{C} = \{(\alpha_1, \alpha_0), (\alpha_2, \alpha_3)\}.$$

□

2.2 Sequence Functions

Now that we have an abstraction for a program, a method of formalizing the order in which the communication actions occur is needed. For $t \geq 0$, let (α_i, t) denote the t^{th} occurrence of the symbol α_i in Γ. Every occurrence of a symbol in a process is assigned a **sequence step**. Within a process, if a given occurrence has to take place before another, then the former will be assigned a sequence step that is numerically less than the one assigned to the latter. To formalize the concept, we make the following definitions:

Definition 2.2 *For a trace* $\mathbf{t}$*, the* **length** *of* $\mathbf{t}$ *is the number of symbols in* $\mathbf{t}$ *and is denoted by* $|\mathbf{t}|$.

Definition 2.3 *For a trace* $\mathbf{t}$ *and a symbol* α_i*, the* **projection** *of* $\mathbf{t}$ *onto* α_i*, denoted by* $\mathbf{t}\lceil\alpha_i$*, is the trace resulting from removing all symbols that are not* α_i *from* $\mathbf{t}$.

Thus, $|\mathbf{t}\lceil\alpha_i|$ yields the number of occurrences of α_i in $\mathbf{t}$. We are now ready to impose ordering on communication actions within a process.

Definition 2.4 *For a program* $\mathcal{P}$*, a* **sequence function** σ *is a function*[1]

$$\sigma : \mathrm{A} \times \mathbf{N} \longrightarrow \mathbf{Z}$$

such that

$$\forall\, \mathbf{t}\alpha_i\alpha_j \in \mathbf{pref}(\Gamma) : \sigma(\alpha_i, |\mathbf{t}\lceil\alpha_i|) < \sigma(\alpha_j, |\mathbf{t}\alpha_i\lceil\alpha_j|). \tag{2.1}$$

For a given set of N processes, $\{p[l] \mid 0 \leq l < N\}$, each implementing the program $\mathcal{P}$, we let $\sigma[l](\alpha_i, t)$ represent the sequence step assigned to the occurrence (α_i, t) in $p[l]$. Clearly, each $\sigma[l]$ should have the property described in the last definition. Moreover, since we are using zero-slack communications, the i^{th} occurrence of α_e in $p[l]$ coincides with the i^{th} occurrence of α_f in $p[l+1]$; so, we require that these two occurrences to be assigned the same sequence step.

[1] **N** is the set of non-negative integers and **Z** is the set of integers.

Definition 2.5 *For a given program $\mathcal{P}$ and a positive integer N, the set of functions $\{\sigma[l] \mid 0 \leq l < N\}$ is a* **consistent set of sequence functions (CSSF)** *if*

$$\forall\, l : 0 \leq l < N : \sigma[l] \text{ is a sequence function for } \mathcal{P}, \tag{2.2}$$

$$\forall\, e, f, l, t : (\alpha_e, \alpha_f) \in \mathrm{C} \wedge 0 < l < N \wedge t \geq 0 : \sigma[l-1](\alpha_e, t) = \sigma[l](\alpha_f, t). \tag{2.3}$$

2.3 Program Properties

Definition 2.6 *A program $\mathcal{P}$ is said to be* **deadlock-free** *if for any positive integer N, there exists a CSSF $\{\sigma[l] \mid 0 \leq l < N\}$.*

The correspondence of this definition with the intuitive notion of what it means for a set of processes to be deadlock-free can be easily demonstrated. First, suppose that there exists a consistent set of sequence functions for program $\mathcal{P}$ and integer N. Next, assume that the linear array of N processes, each implementing $\mathcal{P}$, reaches a deadlock. Of all the "occurrences" that have not been executed by the processes, let (α_i, t) on process $p[l]$ be one with the lowest sequence step. Also, let $(\alpha_{i'}, t')$ on $p[l']$ be the occurrence corresponding to (α_i, t) on $p[l]$. Now, by the definition of consistent set of sequence functions, both these occurrences must have the same sequence step S. Furthermore, by the selection of (α_i, t) and $p[l]$, all occurrences with sequence step less than S must have been executed already. In particular, all occurrences preceding (α_i, t) on $p[l]$ and those preceding $(\alpha_{i'}, t')$ on $p[l']$ have taken place and so the two occurrences can be executed and a deadlock caused by communication dependencies is impossible if there exists a CSSF.

Next, the response time of a set of processes may be interpreted as the latency in time between the sending of an input to the set and the receiving of the associated output. If this time is independent of the total number of processes, then the set is said to have **constant response time**. We modify this definition slightly by requiring that the number of sequence steps separating any two consecutive occurrences be bounded by a constant.

Definition 2.7 *A program $\mathcal{P}$ has* **constant response time (CRT)** *if there exists a constant W, such that for any positive integer N, there exists a CSSF $\{\sigma[l] \mid 0 \leq l < N\}$ with the property that*

$$\forall\, l, \mathbf{t}, i, j : \mathbf{t}\alpha_i\alpha_j \in \mathbf{pref}(\Gamma) \wedge 0 \leq l < N : \sigma[l](\alpha_j, |\mathbf{t}\alpha_i\lceil\alpha_j|) - \sigma[l](\alpha_i, |\mathbf{t}\lceil\alpha_i|) \leq W. \tag{2.4}$$

2.4 Dual Programs

Next, consider the program $\mathcal{P}^{\perp}$ obtained by reversing the order of the indices on the processes in $\mathcal{P}$. We say that $\mathcal{P}^{\perp}$ and $\mathcal{P}$ are duals of each other. It should be clear that the alphabet and communication pattern of the two programs are the same. Moreover, there is a simple relationship between the correspondences as indicated below:

Definition 2.8 *For a program $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$, the* **dual** *program of $\mathcal{P}$ is $\mathcal{P}^{\perp} = \langle \mathrm{A}, \Gamma^{\perp}, \mathrm{C}^{\perp} \rangle$, where*

$$\mathrm{C}^{\perp} = \{(\alpha_f, \alpha_e) \mid (\alpha_e, \alpha_f) \in \mathrm{C}\}$$

and the communication patterns Γ and $\Gamma^{\perp}$ specify the same sequence of symbols.[2]

[2] As will be shown in Chapter 4, for pseudo-cyclic processes, Γ and $\Gamma^{\perp}$ may have different form even though they specify the same communication behavior.

Since a program and its dual basically specify the same communication behavior between processes, it should be clear that $\mathcal{P}$ is deadlock-free (has CRT) if and only if $\mathcal{P}^{\perp}$ is deadlock-free (has CRT).

2.5 Remarks on Correspondences

If $(\alpha_e, \alpha_f), (\alpha_e, \alpha_{f'}) \in \mathrm{C} \wedge f \neq f'$, then a single action in $p[l]$ affects two different actions in $p[l+1]$. This is clearly not possible in our model. Also, if there exist symbols α_i, α_e, α_f, such that $(\alpha_e, \alpha_i), (\alpha_i, \alpha_f) \in \mathrm{C}$, then this would translate into specifying that the execution of a single communication action in $p[l]$ interacts with actions in both $p[l-1]$ and $p[l+1]$. We will ignore communications actions with this property since they do not occur often in practice. So, from now on, we will restrict our attention to situations where the set of correspondences satisfies

$$\forall\, e, f, e', f' : (\alpha_e, \alpha_f), (\alpha_{e'}, \alpha_{f'}) \in \mathrm{C} : e = e' \Leftrightarrow f = f', \tag{2.5}$$

$$\forall\, e, f, e', f' : (\alpha_e, \alpha_f), (\alpha_{e'}, \alpha_{f'}) \in \mathrm{C}; f \neq e'. \tag{2.6}$$

Next, for convenience, the four utility functions below are introduced.

Definition 2.9 *Let* C *be a set of correspondences. Then,*

$$\mathbf{E}(\mathrm{C}) \stackrel{\text{def}}{=} \{e \mid \exists\, f :: (\alpha_e, \alpha_f) \in \mathrm{C}\}; \tag{2.7}$$

$$\mathbf{F}(\mathrm{C}) \stackrel{\text{def}}{=} \{f \mid \exists\, e :: (\alpha_e, \alpha_f) \in \mathrm{C}\}; \tag{2.8}$$

$$\forall\, f : f \in \mathbf{F}(\mathrm{C}) : \mathbf{ecorr}(f, \mathrm{C}) = e \Leftrightarrow (\alpha_e, \alpha_f) \in \mathrm{C}; \tag{2.9}$$

$$\forall\, e : e \in \mathbf{E}(\mathrm{C}) : \mathbf{fcorr}(e, \mathrm{C}) = f \Leftrightarrow (\alpha_e, \alpha_f) \in \mathrm{C}. \tag{2.10}$$

Note that **ecorr**() and **fcorr**() are well defined by (2.5). When the identity of C is obvious from context, the second argument for each of these two functions is often omitted.

Chapter 3

Cyclic Programs

3.1 Definition

Both programs in the examples of Chapter 1 exhibit cyclic communication behavior; namely, the communication pattern of each consists solely of an endless repetition of a fixed sequence of communication actions. This property is formalized by Definition 3.1.

Definition 3.1 *A program* $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ *is said to have* **cyclic communication pattern** *or is* **cyclic** *if* $n = |\mathrm{A}|$ *and*

$$\Gamma = (\alpha_0 \alpha_1 \ldots \alpha_{n-1})^\star.$$

Unless stated otherwise, in this chapter, $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ represents a cyclic program and $n = |\mathrm{A}|$. Also, let $\{p[l] \mid 0 \leq l < N\}$ be a set of N processes, each of which implements $\mathcal{P}$.

Now, because of the regular order in which the symbols occur in Γ, we can determine easily the index of any occurrence; namely, if $\mathbf{t}\alpha_i\alpha_j \in \mathbf{pref}(\Gamma)$, then either

$$(i < n-1 \wedge j = i+1) \vee (i = n-1 \wedge j = 0).$$

Taking advantage of this regularity, we can express (2.1) in the definition of sequence function more specifically as

$$\forall\, i, t : 0 \leq i < n-1 \wedge t \geq 0 : \sigma(\alpha_i, t) < \sigma(\alpha_{i+1}, t), \tag{3.1}$$

$$\forall\, t : t \geq 0 : \sigma(\alpha_{n-1}, t) < \sigma(\alpha_0, t+1). \tag{3.2}$$

3.2 Absence of Deadlocks

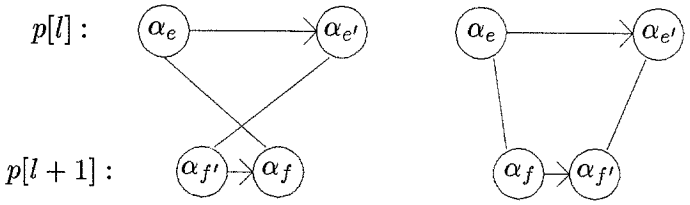
Theorem 1 *A cyclic program* $\mathcal{P}$ *is deadlock-free, if and only if it satisfies the following predicate (called the monotonicity condition):*

$$\forall\, e, f, e', f' : (\alpha_e, \alpha_f), (\alpha_{e'}, \alpha_{f'}) \in \mathrm{C} : e < e' \Leftrightarrow f < f'. \tag{3.3}$$

To see that this requirement is necessary, consider the following example.

Example 3.1: Let $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ represent the following program:

$$*[al?x; bl!(x+y); br?y; ar!x].$$



a. Deadlock b. No Deadlock

Figure 3.1: Dependencies in Cyclic Programs

Using the channel connections as shown in Figure 1.1, C = $\{(\alpha_2, \alpha_1), (\alpha_3, \alpha_0)\}$. Now, suppose $p[0]$ executes the "$br?$" action for the first time. Then, the following dependencies occur: First, the process $p[0]$ cannot complete the "$br?$" action until $p[1]$ has completed its corresponding "$bl!$" action. However, by the topology of the program, $p[1]$ has to complete the "$al?$" action before completing the "$bl!$" action. But, the completion of "$al?$" action by $p[1]$ has to coincide with the the completion of "$ar!$" action in $p[0]$. Since $p[0]$ cannot complete its "$ar!$" action until it has completed the original "$br?$" action, a deadlock results. □

In Figure 3.1a, the orderings between actions when

$$\exists\, e, f, e', f' : (\alpha_e, \alpha_f), (\alpha_{e'}, \alpha_{f'}) \in \mathrm{C} : e < e' \wedge f > f'$$

are displayed graphically. An arrow indicates that the occurrence at the tail must precede the one at the head; since $e < e'$, (α_e, t) must precede $(\alpha_{e'}, t)$ for any t. Furthermore, a correspondence between a pair of occurrences implies that they are to be assigned the same sequence step and is represented by a line segment. The cycle of dependencies in the figure shows that a deadlock occurs. On the other hand, Figure 3.1b depicts the absence of such a cycle when (3.3) is satisfied.

We will next show that the monotonicity condition is also sufficient for a cyclic program to be deadlock-free. By Definition 2.6, given any number of processes N, we need to produce a CSSF $\{\sigma[l] \mid 0 \leq l < N\}$ for $\mathcal{P}$. Toward that end, let an occurrence of the sequence of the n actions $(\alpha_0 \alpha_1 \ldots \alpha_{n-1})$ be called a **phase**. Suppose we can construct a set of sequence functions that are consistent for one phase of the communication pattern; then, due to the periodicity of the communication pattern, we can repeat these functions for subsequent phases. More formally, for any N, we assign to each symbol α_i in process $p[l]$ an integer $\tau[l](i)$ satisfying:

$$\forall\, l, i : 0 \leq l < N \wedge 0 \leq i < n-1 : \tau[l](i) < \tau[l](i+1), \tag{3.4}$$

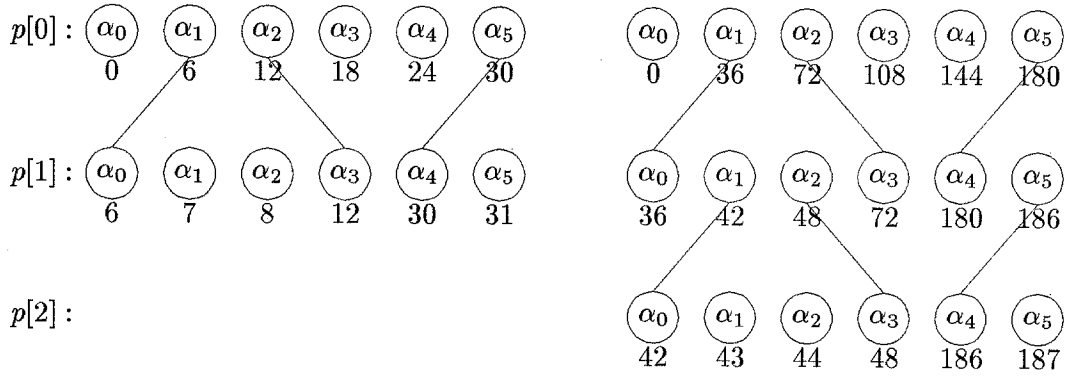
$$\forall\, l, e, f : 0 < l < N \wedge (\alpha_e, \alpha_f) \in \mathrm{C} : \tau[l-1](e) = \tau[l](f). \tag{3.5}$$

Then, observing that the set $\{\tau[l](i)\}$ induces a consistent set of sequence functions for one phase of the communication pattern, we define T_N, the length of each phase, by

$$T_N = \max\{l : 0 \leq l < N : \tau[l](n-1) - \tau[l](0)\} + 1 \tag{3.6}$$

so that there is no overlap between different phases. Then, $\{\sigma[l] \mid 0 \leq l < N\}$ forms a CSSF for $\mathcal{P}$, where

$$\sigma[l](\alpha_i, t) \stackrel{\text{def}}{=} \tau[l](i) + tT_N. \tag{3.7}$$



a. Two Processes b. Three Processes

Figure 3.2: Construction of $\{\tau[l](i)\}$ for Example 3.2

The construction of the set $\{\tau[l](i)\}$ by induction on N is illustrated by the simple example below[1].

Example 3.2: Let $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ where

$$\begin{aligned} \mathrm{A} &= \{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}; \\ \Gamma &= (\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5)^\star; \\ \mathrm{C} &= \{(\alpha_1, \alpha_0), (\alpha_2, \alpha_3), (\alpha_5, \alpha_4)\}. \end{aligned}$$

Figure 3.2 graphically depicts the correspondences. First, note that the monotonicity condition is satisfied. For $N = 1$, we set $\tau[0](i)$ to i and (3.5) will be satisfied vacuously. For $N = 2$, we perform the following steps:

1. For $0 \leq l < N - 1 \wedge 0 \leq i \leq n$, define $\tau[l](i)$ to be n times its value for the case when there are $N - 1$ processes. In our example, this step yields

$$\begin{aligned} &\tau[0](0) = 0, \quad \tau[0](1) = 6, \quad \tau[0](2) = 12, \\ &\tau[0](3) = 18, \quad \tau[0](4) = 24, \quad \tau[0](5) = 30. \end{aligned}$$

2. For each $(\alpha_e, \alpha_f) \in \mathrm{C}$, set $\tau[N-1](f)$ to be $\tau[N-2](e)$. Thus, we have

$$\tau[1](0) = 6, \quad \tau[1](3) = 12, \quad \tau[1](4) = 30.$$

3. For $i \notin \mathbf{F}(\mathrm{C})$, assign to $\tau[N-1](i)$ a value that satisfies (3.4). Since the difference between any pair of $\tau[N-1](f)$ and $\tau[N-1](f')$ defined in the previous step must be at least n and there are at most n symbols, this assignment can always be done.

   For our example, the procedure would yield

$$\tau[1](1) = 7, \quad \tau[1](2) = 8, \quad \tau[1](5) = 31.$$

[1]The construction only serves to prove the validity of Theorem 1, its efficiency is irrelevant

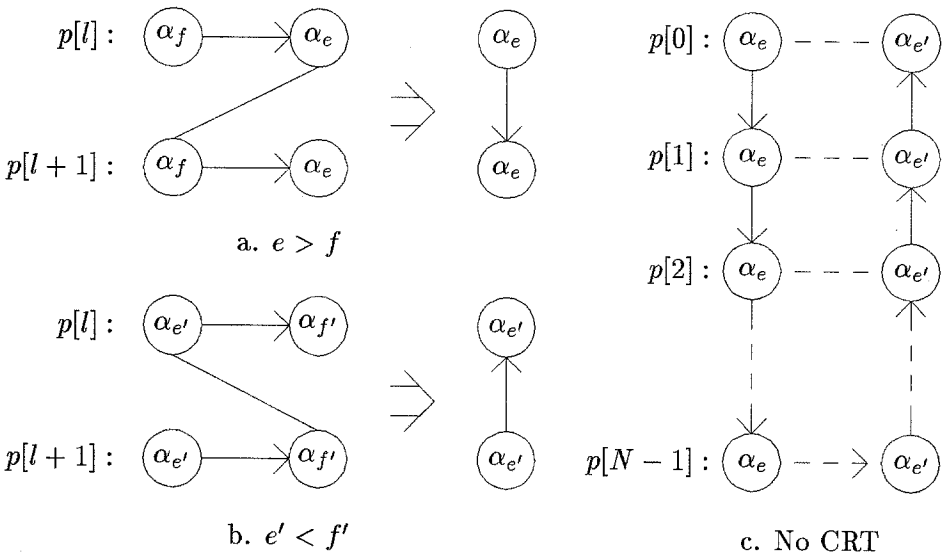


Figure 3.3: Violation of Uni-direction Condition

The resultant assignment is shown in Figure 3.2a. Clearly, by repeating the above procedure, $\{\tau[l](i)\}$ can be constructed for any number of processes. The situation when there are three processes is shown in Figure 3.2b.

For the case when there are two processes, $T_N = 31$ and $\{\sigma[0], \sigma[1]\}$ defined by

$$\begin{array}{lll} \sigma[0](\alpha_0, t) = 0 + 31t; & \sigma[0](\alpha_1, t) = 6 + 31t; & \sigma[0](\alpha_2, t) = 12 + 31t; \\ \sigma[0](\alpha_3, t) = 18 + 31t; & \sigma[0](\alpha_4, t) = 24 + 31t; & \sigma[0](\alpha_5, t) = 30 + 31t; \\ \sigma[1](\alpha_0, t) = 6 + 31t; & \sigma[1](\alpha_1, t) = 7 + 31t; & \sigma[1](\alpha_2, t) = 8 + 31t; \\ \sigma[1](\alpha_3, t) = 12 + 31t; & \sigma[1](\alpha_4, t) = 30 + 31t; & \sigma[1](\alpha_5, t) = 31 + 31t; \end{array}$$

is a CSSF for $\mathcal{P}$. □

The formal construction and verification of $\{\tau[l](i)\}$ and $\{\sigma[l]\}$ are straightforward and are not included.

3.3 Constant Response Time

Theorem 2 *A program $\mathcal{P}$ has CRT if and only if it satisfies the monotonicity condition and the uni-direction condition below:*

$$(\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C} : e > f) \vee (\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C} : e < f). \tag{3.8}$$

The necessity of the monotonicity condition should be obvious. Next, suppose there is a correspondence (α_e, α_f) such that $e > f$ as shown in Figure 3.3a. Since (α_e, t) in $p[l]$ coincides with (α_f, t) in $p[l+1]$, (α_e, t) in $p[l]$ must precede (α_e, t) in $p[l+1]$ by at least one step. We'll indicate this precedence by a direct arrow as shown on the right side of the figure. By repeating this relationship, (α_e, t) in $p[0]$ must precede (α_e, t) in $p[N-1]$ by at least $N-1$ steps. See Figure 3.3c where each arrow implies a precedence of at least one step.

Symmetrically, if $(\alpha_{e'}, \alpha_{f'}) \in \mathrm{C}$ and $e' < f'$, then Figure 3.3b indicates that $(\alpha_{e'}, t)$ in $p[l]$ must *follow* $(\alpha_{e'}, t)$ in $p[l+1]$ by at least one step. Thus, $(\alpha_{e'}, t)$ in $p[N-1]$ must

precede $(\alpha_{e'}, t)$ in $p[0]$ by at least $N-1$ steps. Since there must be an occurrence of $\alpha_{e'}$ after an occurrence of α_e in $p[0]$, it should be clear from Figure 3.3c that the number of steps between these two occurrences must be at least $2N-2$ and cannot be bounded above by a constant. Therefore, $\mathcal{P}$ cannot have CRT. The argument can be formalized[2] to show the necessity of the uni-direction condition for a program with CRT.

To establish the sufficiency of (3.3) and (3.8), we attempt to produce a constant M and a set of integers $\{\pi(i) \mid 0 \leq i < n\}$ such that

$$\forall\, i : 0 \leq i < n-1 : \pi(i) < \pi(i+1), \tag{3.9}$$

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C} : \pi(e) = \pi(f) + M. \tag{3.10}$$

Once these values are obtained, we let

$$T = \pi(n-1) - \pi(0) + 1 \tag{3.11}$$

be the length of each phase and then, for any N, $\{\sigma[l] \mid 0 \leq l < N\}$ defined as

$$\sigma[l](\alpha_i, t) \stackrel{\text{def}}{=} \pi(i) + lM + tT \tag{3.12}$$

will be a CSSF for $\mathcal{P}$. Moreover, since the difference in sequence steps between any two consecutive occurrences must be less than T, $\mathcal{P}$ has CRT by (2.4).

The actual construction of M and $\{\pi(i) \mid 0 \leq i < n\}$ when $\mathcal{P}$ satisfies

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C} : f < e \tag{3.13}$$

is performed by Algorithm 1 and associated Procedure 1A described in the Appendix. Note that if, instead, $\mathcal{P}$ satisfies

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C} : f > e \tag{3.14}$$

then we can replace $\mathcal{P}$ with its dual, which has the same communication behavior as $\mathcal{P}$ and satisfies (3.13).

Since graphical representations tend to make Algorithm 1 easier to follow, we define the "correspondence graph" of a program $\mathcal{P}$, denoted by $\mathrm{G}(\mathcal{P})$, as the directed graph $\langle \mathrm{V}, \mathrm{E} \rangle$ where

$$\mathrm{V} = \{i \mid \alpha_i \in \mathrm{A}\},$$
$$\mathrm{E} = \{(e, f) \mid (\alpha_e, \alpha_f) \in \mathrm{C}\}.$$

For each edge (e, f) in $\mathrm{G}(\mathcal{P})$, e is the called the "tail" vertex of the edge and f is the "head" vertex. By convention, correspondence graphs are drawn with vertices lying on the same horizontal row and numbered from left to right in ascending order. Note that a program satisfying the monotonicity condition means that its correspondence graph does not have any of the subgraphs shown in Figure 3.4. Also, a program satisfying (3.13) means that the edges of its correspondence graph are all pointing to the left.

For reference, let $\pi(i)$ be called the "π-value" of vertex i. Also, for each edge (e, f), let the $\pi(e) - \pi(f)$ be referred to as the "π-difference" of that edge. Algorithm 1 systematically

[2] Basically, the proof consists of replacing the precedences described here by inequalities on the sequence steps of the relevant occurrences and then summing the inequalities to show that the response time depends on N.

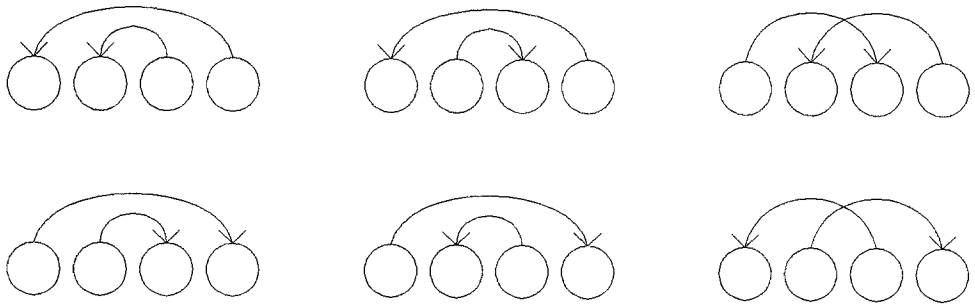


Figure 3.4: Subgraphs Indicating Presence of Deadlocks

defines $\pi(k)$ as k ranges from 0 to $n-1$. It attempts to construct $\{\pi(i) \mid 0 \leq i < n\}$ by enforcing the proper ordering within $\{\pi(i) \mid 0 \leq i \leq k\}$ and maintaining the following loop invariant:

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C} \wedge e \leq k : \pi(e) - \pi(f) = M. \tag{3.15}$$

Graphically, this invariant translates into requiring that all the edges not to the right[3] of vertex k have the same π-difference of M.

Below is the rephrasing of Algorithm 1 in more intuitive terms; its application to a general example and argument for its correctness follow immediately.

ALGORITHM 1:
Input: Cyclic program $\mathcal{P}$ satisfying (3.3), (3.8), and (3.13).
Output: Integer M and set of integers $\{\pi(i) \mid 0 \leq i < n\}$ satisfying (3.9) and (3.10).

1. Initialize M, $\pi(0)$, and k to 0.
2. If $k = n-1$, then exit algorithm.
3. Increment k and set $\pi(k) = \pi(k-1) + 1$.
4. If k is not a tail vertex then go on to the next step. If k is a tail vertex, let k' be the corresponding head vertex. If the π-difference of (k, k') is not greater than M, then increase $\pi(k)$, if necessary, so as to maintain (3.15). If the π-difference of (k, k') is greater than M, then let $\delta = \pi(k) - \pi(k') - M$. Next, execute Procedure 1A so as to increase the π-differences for all edges to the left of vertex k by δ in order to maintain (3.15). During the application of the procedure, all edges to the right of vertex k are ignored.
5. Go to Step 2.

PROCEDURE 1A:

1. Initialize x to $k-1$.
2. Find $\hat{e}$, the rightmost tail vertex not to the right of vertex x. If no such vertex exists, then set M to $M + \delta$ and exit procedure.

[3] An edge (e, f) is to the right of vertex k if $e > k$.

3. Let $\hat{f}$ be the head vertex corresponding to $\hat{e}$, and increase the π-values for all vertices to the right of $\hat{f}$ by δ.

4. Set x to $\hat{f}$ and go to Step 2.

Example 3.3: Let $\mathcal{P} = (\mathrm{A}, \Gamma, \mathrm{C})$ with

$$\begin{aligned} \mathrm{A} &= \{\alpha_0, \alpha_1, \ldots, \alpha_9\}, \\ \Gamma &= (\alpha_0\alpha_1 \ldots \alpha_9)^\star, \\ \mathrm{C} &= \{(\alpha_2, \alpha_0), (\alpha_5, \alpha_1), (\alpha_6, \alpha_3), (\alpha_8, \alpha_4), (\alpha_9, \alpha_7)\}. \end{aligned}$$

The correspondence graph of $\mathcal{P}$ is shown in Figure 3.5a. The operations performed by Algorithm 1 are shown below and illustrated in the rest of Figure 3.5 with the number under each node representing its *pi*-value.

1. Initially, $M = 0$ and $\pi(0) = 0$ and $k = 0$.

2. $\pi(1)$ becomes 1. The invariant is maintained since vertex 1 is not a tail vertex.

3. $\pi(2)$ becomes 2. Since vertex 2 is a tail vertex, we identify vertex 0 as its corresponding head vertex. Next, since $\pi(2) - \pi(0) = 2 - 0$ which is greater than $M = 0$, Procedure 1A is to be applied. However, since this is the first correspondence encountered, the procedure exits immediately after setting M to 2, thereby maintaining (3.15).

4. $\pi(3)$ becomes 3.

5. $\pi(4)$ becomes 4.

6. $\pi(5)$ becomes 5; see Figure 3.5b. This time, $\pi(5) - \pi(1) = 4 > M$, so we set $\delta = \pi(5) - \pi(1) - M = 2$. Next, Procedure 1A is applied to increase, by δ, the π-differences for all edges to the left of vertex 5. First, the rightmost tail vertex to the left of vertex 5 is found. This is vertex 2. We then identify its corresponding head vertex as vertex 0 and add δ to each $\pi(i)$ with $i > 0$. The resultant configuration is shown in Figure 3.5c. In Figure 3.5b, the increments by δ are indicated by placing a δ symbol between vertices 0 and 1 — the interpretation is that vertices to the right of m δ symbols have their π-values increased by $m\delta$. Note that the placement of a δ symbol between the tail and the head of an edge effectively increases the π-difference for that edge by δ. Such an edge is said to have been "stretched". Thus, by placing δ between vertices 0 and 1, $(\pi(2) - \pi(0))$ is increased from 2 to 4 while $(\pi(5) - \pi(1))$ is left unchanged. Thus, after the modifications, (3.15) holds with $M = 4$.

7. $\pi(6)$ becomes $\pi(5) + 1 = 8$. But $\pi(6) - \pi(3) = 3 < M$; so $\pi(6)$ is incremented to 9 to maintain (3.15).

8. $\pi(7)$ becomes $\pi(6) + 1 = 10$.

9. $\pi(8)$ becomes $\pi(7) + 1 = 11$. Since $\pi(8) - \pi(4) = 5 > M$, Procedure 1A has to be applied again with $\delta = 1$. The rightmost tail vertex to the left of vertex 8 is vertex 6. The corresponding head vertex is vertex 3, so a δ symbol is placed between vertices 3

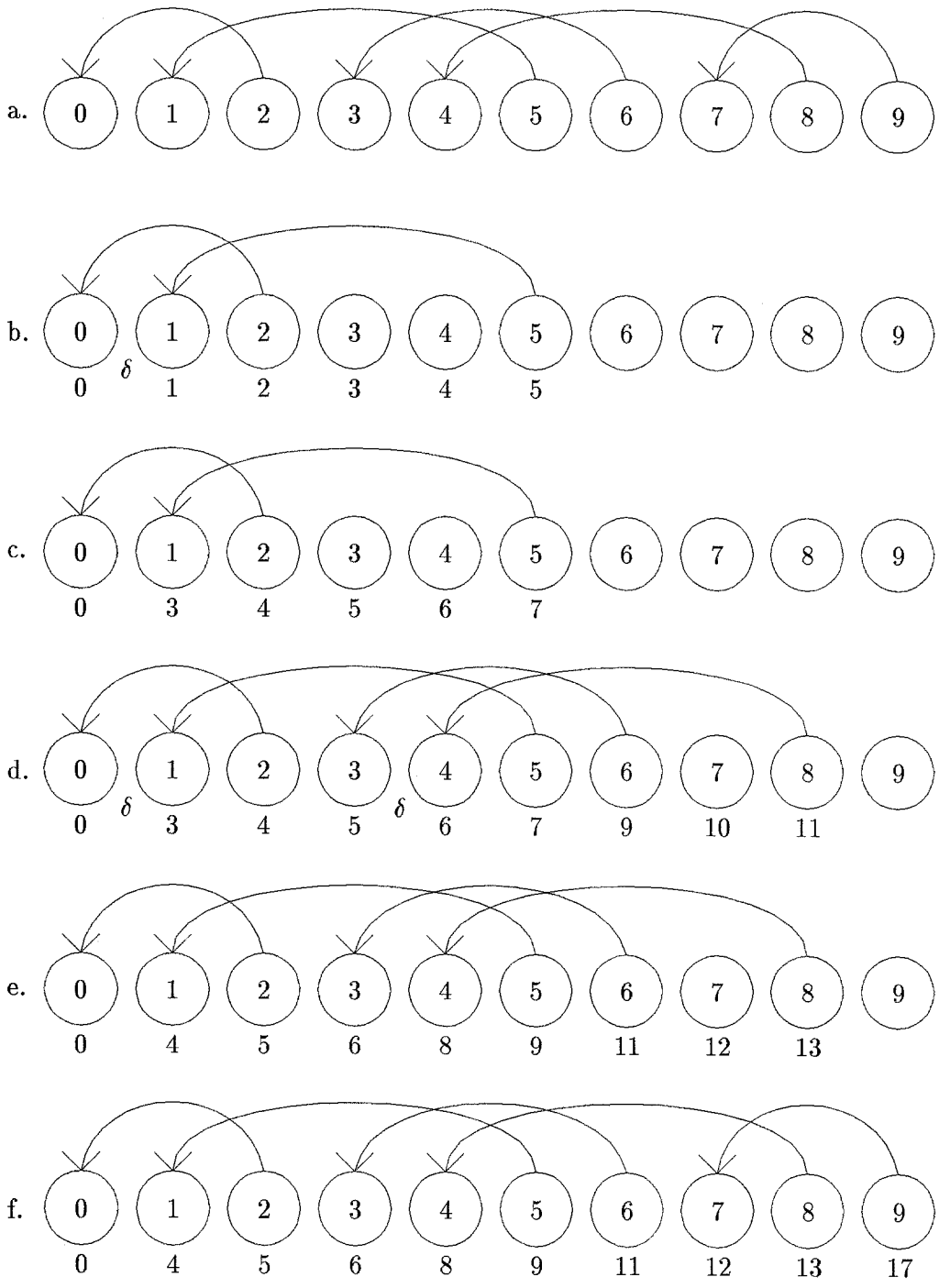


Figure 3.5: Application of Algorithm 1 for Example 3.3

and 4. See Figure 3.5d. Next, the rightmost tail vertex to the left of 3 is found — this is vertex 2. Again, the corresponding head vertex is identified (vertex 0 in this case) and a δ symbol is placed immediately to the right of it. The resultant configuration is shown in Figure 3.5e. Again, notice the strategic placement of exactly one δ symbol between the head and tail of all edges to the left of vertex 8 in Figure 3.5d.

10. $\pi(9)$ becomes $\pi(8) + 1 = 14$. But $\pi(9) - \pi(7) = 2 < M$; so $\pi(9)$ is incremented to 17 to maintain (3.15). See the final configuration in Figure 3.5f.

Once M and $\{\pi(i) \mid 0 \leq i < n\}$ have been constructed, it is straightforward to define $\{\sigma[l] \mid 0 \leq l < N\}$ by

$$\begin{array}{ll} \sigma[l](\alpha_0, t) = 0 + 18t + 5l, & \sigma[l](\alpha_5, t) = 9 + 18t + 5l, \\ \sigma[l](\alpha_1, t) = 4 + 18t + 5l, & \sigma[l](\alpha_6, t) = 11 + 18t + 5l, \\ \sigma[l](\alpha_2, t) = 5 + 18t + 5l, & \sigma[l](\alpha_7, t) = 12 + 18t + 5l, \\ \sigma[l](\alpha_3, t) = 6 + 18t + 5l, & \sigma[l](\alpha_8, t) = 13 + 18t + 5l, \\ \sigma[l](\alpha_4, t) = 8 + 18t + 5l, & \sigma[l](\alpha_9, t) = 17 + 18t + 5l, \end{array}$$

and verify that it is a CSSF for $\mathcal{P}$, and that it induces CRT. □

The purpose of the loop in Procedure 1A is to perform δ placement so that all edges not to the right of vertex k have the same π-difference of $M + \delta$. It accomplishes this by maintaining the following predicate as a precondition of Step 2:

$$\begin{aligned} \forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C} : (e \leq x \Rightarrow \pi(e) - \pi(f) = M) \wedge \\ (x < e \leq k \Rightarrow \pi(e) - \pi(f) = M + \delta). \end{aligned} \tag{3.16}$$

Initially, x is $k - 1$; so, by the state of the variables when this procedure is called in Algorithm 1, the predicate is established. Suppose that 3.16 holds prior to the i^{th} execution of Step 2 in the procedure. The increase of π-values in Step 3 is equivalent to placing a δ to the immediate right of vertex $\hat{f}$, thereby stretching all edges (e, f) satisfying

$$f \leq \hat{f} < e.$$

But by the monotonicity condition, this implies $e \leq \hat{e}$. Furthermore, by the choice of $\hat{e}$, there is no tail vertex e such that $\hat{e} < e \leq x$. Thus, an edge (e, f) has its π-value increased by δ if and only if $\hat{f} < e \leq x$. The assignment of $\hat{f}$ to x at Step 4 then establishes our claim.

Since Procedure 1A terminates when there are no more edges to the right of x, the setting of M to $M + \delta$ maintains invariant 3.15 of Algorithm 1. Also, note that the ordering among the $\pi(i)$'s is preserved by their initial assignments and the fact that they are modified only by δ placements. It then follows that the algorithm produces the appropriate values for $\{\pi(i) \mid 0 \leq i < n\}$ and M. Thus, a program satisfying the monotonicity and uni-direction conditions has CRT.

3.4 Concluding Remarks

We now have simple criteria for characterizing cyclic programs. Also, though the actual construction of the CSSF may take $O(|\mathrm{C}|^3)$, the checking of whether a program satisfies these conditions is straightforward and takes at most $O(|\mathrm{C}| \log |\mathrm{C}|)$ for sorting the correspondences.

Chapter 4

Pseudo-Cyclic Programs

4.1 Definition

In this chapter, we will investigate the behavior of programs with "pseudo-cyclic" communication patterns; that is, patterns that are cyclic except for some leading symbols. These programs are fairly common due to the need of communications to initialize variables; the following is one such example.

Example 4.1: Consider the following program for recognizing the occurrence of a given substring:

$$al?z; xold := z; *[bl?y; al?x; ar!x; br!((z = xold) \wedge y); xold := x]$$

Again, the connection shown in Figure 1.1 is used, and the environment is assumed to supply the constant value "**true**" on the "*br*!" channel. Then, for the three processes, each implementing this program, the following computation is performed:

$$b[i] \equiv (a[0] = a[i]) \wedge (a[1] = a[i+1]) \wedge (a[2] = a[i+2]),$$

where $a[i]$ is the i^{th} input value of the "*al*?" action in $p[0]$ and $b[i]$ is the i^{th} output value of the "*br*!" action in $p[N-1]$. The need to initially load in the substring $a[0]a[1]a[2]$ requires a non-cyclic communication pattern. This program can be represented by $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$, where

$$\begin{aligned} \mathrm{A} &= \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}, \\ \Gamma &= \alpha_1(\alpha_0\alpha_1\alpha_2\alpha_3)^\star, \\ \mathrm{C} &= \{(\gamma_2, \gamma_1), (\gamma_3, \gamma_0)\}. \end{aligned}$$

□

Definition 4.1 *A program $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ is said to have* **pseudo-cyclic communication behavior** *or is* **pseudo-cyclic** *if Γ is of the form*

$$\mathrm{a}_0\mathrm{a}_1 \ldots \mathrm{a}_{n'-1}(\alpha_0\alpha_1 \ldots \alpha_{n-1})^\star,$$

where $\mathrm{A} = \{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$ and for $0 \leq i < n', \mathrm{a}_i \in \mathrm{A}$. The string $\mathbf{a} = \mathrm{a}_0\mathrm{a}_1 \ldots \mathrm{a}_{n'-1}$ is called the **head** *of the communication pattern; the non-terminating sequence of symbols $(\alpha_0\alpha_1 \ldots \alpha_{n-1})^\star$ is called the* **body** *of the communication pattern.*

Unless specified otherwise, in this chapter, $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ represents a program with pseudo-cyclic communication pattern. Also, n is the size of A; $\mathbf{a}$ is the head of Γ; and n' is the length of $\mathbf{a}$. Moreover, let $\{p[l] \mid 0 \leq l < N\}$ represent a set of N processes, each implementing $\mathcal{P}$. Also, for ease of reference, let a_i be the i^{th} symbol in Γ.

Ignoring the trivial case where $\mathrm{C} = \emptyset$, we define

$$g \stackrel{\text{def}}{=} \min\{i \mid \exists\, j :: (\alpha_i, \alpha_j) \in \mathrm{C}\}. \tag{4.1}$$

We also let α_h denote the corresponding action of αg; *i.e.*, $(\alpha g, \alpha_h) \in \mathrm{C}$.

Example 4.2: For the substring recognizer example, $g = 2$ and $h = 1$. □

4.2 Lags, Event Numbers, and Phase Differences

Definition 4.2 *For each symbol $\alpha_i \in \mathrm{A}$, the* **lag** *of α_i is*

$$\mathbf{lag}(\alpha_i) \stackrel{\text{def}}{=} |\mathbf{a}\lceil\alpha_i|. \tag{4.2}$$

Example 4.3: In the preceding example, $\mathbf{lag}(\alpha_0) = \mathbf{lag}(\alpha_2) = \mathbf{lag}(\alpha_3) = 0$, and $\mathbf{lag}(\alpha_1) = 1$. □

Note that the first occurrence of α_i in the body of Γ must be $(\alpha_i, \mathbf{lag}(\alpha_i))$, since there are exactly $\mathbf{lag}(\alpha_i)$ occurrences of α_i in $\mathbf{a}$. In fact, if we adapt the term "phase" to denote the occurrence of the sequence of actions $(\alpha_0\alpha_1 \ldots \alpha_{n-1})$ in the body of Γ, then the occurrence of α_i in phase t is precisely $(\alpha_i, t + \mathbf{lag}(\alpha_i))$. Thus, even though occurrences of actions in the head can be arbitrary, those in the body follow a regular pattern that can be described by the lemma below.

Lemma 4.1 *Let σ be a sequence function for $\mathcal{P}$, then*

$$\forall\, i, t : 0 \leq i < n-1 \wedge t \geq 0 : \quad \sigma(\alpha_i, t + \mathbf{lag}(\alpha_i)) < \sigma(\alpha_{i+1}, t + \mathbf{lag}(\alpha_{i+1})), \tag{4.3}$$

$$\forall\, i, j, t : \alpha_i, \alpha_j \in \mathrm{A} \wedge t \geq 0 : \quad \sigma(\alpha_i, t + \mathbf{lag}(\alpha_i)) < \sigma(\alpha_j, t + 1 + \mathbf{lag}(\alpha_j)). \tag{4.4}$$

The predicate (4.3) describes the sequencing of occurrences within a given phase and (4.4) describes the ordering of occurrences between different phases.

A concept that is useful for referencing a particular occurrence is its "event number". Intuitively, the event number of an occurrence is the index of the symbol in Γ that represents that occurrence. Recalling that a_i represents the i^{th} symbol in Γ, we have

Definition 4.3 *For a program $\mathcal{P}$, the* **event number** *of an occurrence (α_i, t) is defined by*

$$\mathbf{ev}(\alpha_i, t) = v \Leftrightarrow (\mathrm{a}_v = \alpha_i \wedge |\mathrm{a}_0\mathrm{a}_1 \ldots \mathrm{a}_{v-1}\lceil\alpha_i| = t).$$

Example 4.4: Continuing with the previous example, $\mathbf{ev}(\alpha_1, 0) = 0$, $\mathbf{ev}(\alpha_1, 1) = 2$, $\mathbf{ev}(\alpha_1, 2) = 6$, and $\mathbf{ev}(\alpha_1, 3) = 10$. □

If σ is a sequence function for $\mathcal{P}$, then it should be clear from Def. 2.4 that

$$\mathbf{ev}(\alpha_i, t) < \mathbf{ev}(\alpha_{i'}, t') \Leftrightarrow \sigma(\alpha_i, t) < \sigma(\alpha_{i'}, t'). \tag{4.5}$$

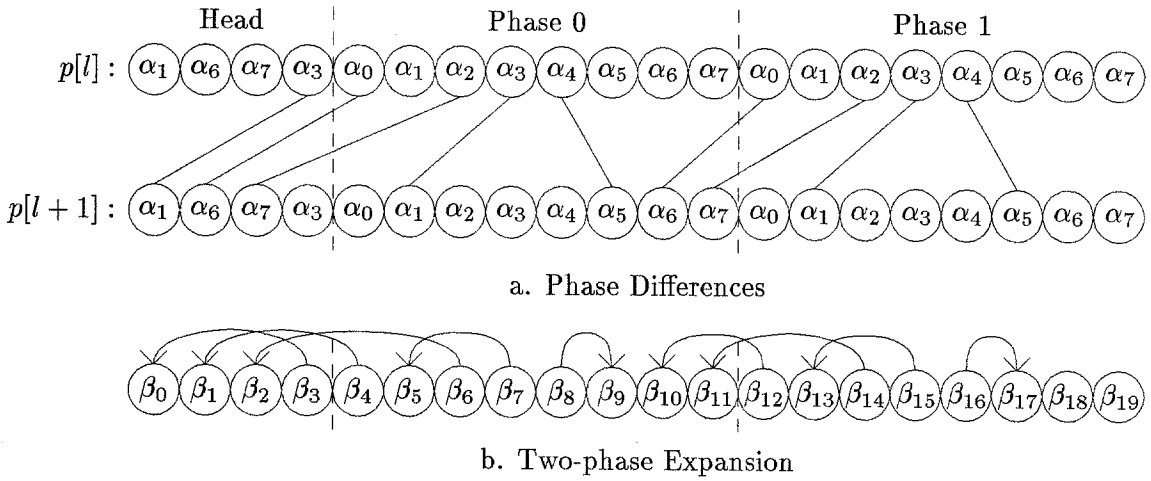


Figure 4.1: Correspondences in Example 4.5

Also, it can be verified without much difficulty that

$$\mathbf{ev}(\alpha_i, t) < n' \;\Rightarrow\; t < \mathbf{lag}(\alpha_i), \tag{4.6}$$

$$\mathbf{ev}(\alpha_i, t) \geq n' \;\Rightarrow\; \mathbf{ev}(\alpha_i, t) = n' + i + (t - \mathbf{lag}(\alpha_i))n. \tag{4.7}$$

Example 4.5: Consider pseudo-cyclic program $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ with

$$\begin{aligned} \mathrm{A} &= \{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7\}; \\ \Gamma &= \alpha_1\alpha_6\alpha_7\alpha_3(\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7)^\star; \\ \mathrm{C} &= \{(\alpha_0, \alpha_6), (\alpha_2, \alpha_7), (\alpha_3, \alpha_1), (\alpha_4, \alpha_5)\}. \end{aligned}$$

Figure 4.1a shows the correspondences between the occurrences of symbols in $p[l]$ and $p[l+1]$.

Note that in this example, the occurrence of α_0 in one phase of $p[l]$ coincides with the occurrence of α_6 in a different phase of $p[l+1]$. This "phase difference" between corresponding occurrences arises because α_6 appears in the head of Γ and α_0 does not; as a result, α_0 in $p[l]$ is always one phase ahead of the corresponding α_6 in $p[l+1]$. Furthermore, note that for the correspondence (α_3, α_1), there is no phase difference because the head of Γ contains the same number of α_3 as α_1. □

Definition 4.4 *For each* $(\alpha_e, \alpha_f) \in \mathrm{C}$, *the* **phase difference** *of the correspondence is*

$$\Delta(\alpha_e, \alpha_f) \stackrel{\text{def}}{=} \mathbf{lag}(\alpha_f) - \mathbf{lag}(\alpha_e). \tag{4.8}$$

Example 4.6: In the previous example, $\Delta(\alpha_0, \alpha_6) = 1 - 0 = 1$, $\Delta(\alpha_2, \alpha_7) = 1 - 0 = 1$, $\Delta(\alpha_3, \alpha_1) = 1 - 1 = 0$, and $\Delta(\alpha_4, \alpha_5) = 0 - 0 = 0$. □

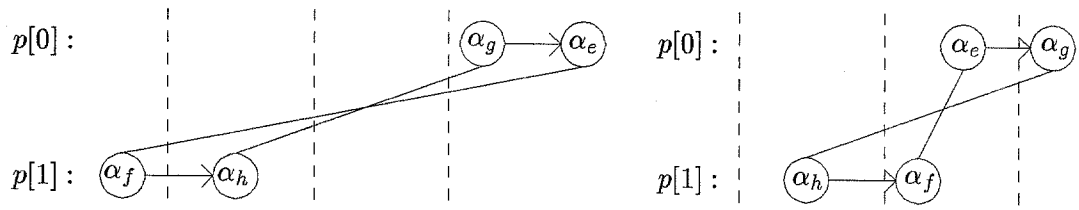


Figure 4.2: Deadlock Across Phases

4.3 Canonical Programs

The following lemma restricts the phase differences that the correspondences in a given deadlock-free program may have.

Lemma 4.2 *If a program $\mathcal{P}$ is deadlock-free, then*

$$\forall\, e,f : (\alpha_e,\alpha_f) \in \mathrm{C} : \Delta(\alpha_g,\alpha_h) - 1 \leq \Delta(\alpha_e,\alpha_f) \leq \Delta(\alpha_g,\alpha_h). \tag{4.9}$$

Figure 4.2 serves to illustrate the validity of this lemma. Suppose that the phase difference between α_g and its corresponding action α_h is 2, as shown in Figure 4.2a where the vertical dashed lines separate consecutive phases. Then, a phase difference of greater than 2 for the correspondence (α_e,α_f) would create a cycle of dependencies analogous to the one in Figure 3.1 of Section 3.2. Similarly, a phase difference of zero or less for (α_e,α_f) would also cause a deadlock, as shown in Figure 4.2b.

Definition 4.5 *For a program $\mathcal{P}$, the* **maximum phase difference** *is $\Delta(\alpha_g,\alpha_h)$.*

Definition 4.6 *A program $\mathcal{P}$ is called* **canonical** *if its correspondences satisfy (4.9) and its maximum phase difference is 1.*

For most of the remainder of this chapter, only canonical programs will be discussed. However, in the last section of the chapter, there will be descriptions on how to transform any pseudo-cyclic program that satisfies (4.9) into a canonical program.

4.4 Absence of Deadlocks

Due to the arbitrary nature of **a**, the head of the communication pattern, it becomes useful to explicitly "trace out" the initial occurrences of symbols in Γ and their correspondences. This leads to the following definition.

Definition 4.7 *For any integer $m > 0$, the m-step* **expansion** *of a program $\mathcal{P}$ is another program $\bar{\mathcal{P}} = \langle \bar{\mathrm{A}}, \bar{\Gamma}, \bar{\mathrm{C}} \rangle$ where*

- $\bar{\mathrm{A}} = \{\beta_i \mid 0 \leq i < m\}$;
- $\bar{\Gamma} = \beta_0\beta_1 \ldots \beta_{m-1}$;
- $\bar{\mathrm{C}} = \{(\beta_{e'},\beta_{f'}) \mid \exists\ e,f,t : (\alpha_e,\alpha_f) \in \mathrm{C} \wedge t \geq 0 :$
  $e' = \mathbf{ev}(\alpha_e,t) \wedge f' = \mathbf{ev}(\alpha_f,t) \wedge e' < m \wedge f' < m\}$.

The $(n' + 2n)$-step expansion of $\mathcal{P}$ is also called the **two-phase expansion** *of $\mathcal{P}$.*

Thus, the set $\bar{\mathrm{C}}$ is defined as including all correspondences that occur within the first m occurrences of symbols.

Example 4.7: The set of correspondences of the two-phase expansion of the program in Example 4.5 is

$$\bar{\mathrm{C}} = \{(\beta_3, \beta_0), (\beta_4, \beta_1), (\beta_6, \beta_2), (\beta_7, \beta_5), (\beta_8, \beta_9), (\beta_{12}, \beta_{10}), (\beta_{14}, \beta_{11}), (\beta_{15}, \beta_{13}), (\beta_{16}, \beta_{17})\}.$$

The corresponding graph of this expansion is shown in Figure 4.1b. Note that $\{\beta_i \mid n' + n(t-1) \leq i < n' + nt\}$ represents the occurrences in phase t of Γ. □

Theorem 3 *Let $\mathcal{P}$ be a canonical program $\mathcal{P}$ and $\bar{\mathcal{P}} = \langle \bar{\mathrm{A}}, \bar{\Gamma}, \bar{\mathrm{C}} \rangle$ be its two-phase expansion. Then, $\mathcal{P}$ is deadlock-free if and only if $\bar{\mathcal{P}}$ satisfies the monotonicity condition below*

$$\forall\, e, f, e', f' : (\beta_e, \beta_f), (\beta_{e'}, \beta_{f'}) \in \bar{\mathrm{C}} : e < e' \Leftrightarrow f < f'. \tag{4.10}$$

The necessity of this condition should be obvious because otherwise there would be a cycle of dependencies in the expansion and deadlock would result. The proof is analogous to that for showing the necessity of the monotonicity condition for deadlock-free cyclic programs.

The argument to show that this condition is also sufficient is similar to that for Theorem 1 in Chapter 3. However, because of the interaction between consecutive phases, we choose to construct sequence functions that are consistent for the head and the first *two* phases[1] of the communication pattern and to maintain a constant difference T_N between matching actions in these two phases. More precisely, we are looking for T_N and a set of integers $\{\tau[l](i) \mid 0 \leq l < N \wedge 0 \leq i < n' + 2n\}$ such that

$$\forall\, l, i : 0 \leq l < N \wedge 0 \leq i < n' + 2n - 1 : \tau[l](i) < \tau[l](i+1), \tag{4.11}$$

$$\forall\, l, e, f : 0 < l < N \wedge (\beta_e, \beta_f) \in \bar{\mathrm{C}} : \tau[l-1](e) = \tau[l](f), \tag{4.12}$$

$$\forall\, l, i : 0 \leq l < N \wedge n' \leq i < n' + n : \tau[l](i+n) = \tau[l](i) + T_N. \tag{4.13}$$

Once $\{\tau[l](i)\}$ has been constructed, we need to produce the set of sequence functions for $\mathcal{P}$. If v, the event number of (α_i, t) is less than $n' + n$, then that occurrence has been included in the two-phase expansion and so its corresponding $\tau[l](v)$ value will be assigned to $\sigma[l](\alpha_i, t)$. If the event number of the occurrence is greater than $n' + n$, then extrapolation using T_N as the length of each phase will be employed. More precisely, using the relationships in (4.6) and (4.7), we can define $\{\sigma[l] \mid 0 \leq l < N\}$ as

$$\sigma[l](\alpha_i, t) = \begin{cases} \tau[l](\mathbf{ev}(\alpha_i, t)) & \text{if } t < \mathbf{lag}(\alpha_i) \\ \tau[l](n' + i) + (t - \mathbf{lag}(\alpha_i))T_N & \text{if } t \geq \mathbf{lag}(\alpha_i), \end{cases} \tag{4.14}$$

and verify that it is a CSSF for $\mathcal{P}$.

Example 4.8: To illustrate the algorithm for generating T_N and $\{\tau[l](i)\}$, consider again the two-phase expansion of Example 4.5. If $N = 1$, then set $\tau[0](i)$ to i and T_1 to n. For $N = 2$, we perform the following steps:

[1] Less steps may be sufficient but to simplify the proof, $n' + 2n$ steps are used.

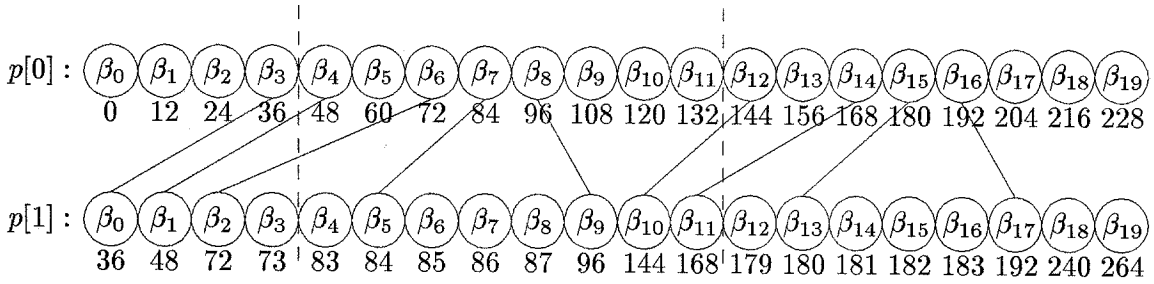


Figure 4.3: Construction of $\{\tau[l](i)\}$ for Example 4.8

1. Let $K = n + n'$. In our example, $K = 12$.

2. For $0 \leq l < N - 1 \wedge 0 \leq i < n' + 2n$, define $\tau[l](i)$ to be K times its value for the case when there are $N - 1$ processes. In our example, this step yields

$$\begin{array}{lll} \tau[0](0) = 0, & \tau[0](1) = 12, & \tau[0](2) = 24, \\ \tau[0](3) = 36, & \tau[0](4) = 48, & \ldots \end{array}$$

   Also, T_N becomes K times T_{N-1}. So, $T_2 = 12 \times 8 = 96$ in the example.

3. For each $(\beta_e, \beta_f) \in \bar{\mathrm{C}}$, set $\tau[N-1](f)$ to be $\tau[N-2](e)$. Thus, as shown in Figure 4.3, we have the following assignments for our example:

$$\begin{array}{lll} \tau[1](0) = 36, & \tau[1](1) = 48, & \tau[1](2) = 72, \\ \tau[1](5) = 84, & \tau[1](9) = 96, & \tau[1](10) = 144, \\ \tau[1](11) = 168, & \tau[1](13) = 180, & \tau[1](17) = 192. \end{array}$$

4. For the remaining actions in the head and phase 0 of Γ, assign to $\tau[N-1](i)$ a value that satisfies (4.11). Since the difference between any pair of $\tau[N-1](f)$ and $\tau[N-1](f')$ in the previous step must be at least K, and since there are at most K symbols with indices less than $n' + n = K$, this assignment can always be done.

   However, an additional criterion has to be satisfied by the assignments; namely,

$$\tau[N-1](n' + n - 1) - \tau[N-1](n') \leq T_N$$

   since T_N is the length of the phase. This can be accomplished easily by assigning the maximum possible values to the leftmost set of actions in phase 0 ($\{\beta_4\}$ in the example) and the minimum possible values to all other ones.

   For our example, the following assignments are made:

$$\begin{array}{lll} \tau[1](3) = 73, & \tau[1](4) = 83, & \tau[1](6) = 85, \\ \tau[1](7) = 86, & \tau[1](8) = 87. & \end{array}$$

5. For all actions β_i in the second phase, let $\tau[N-1](i) = \tau[N-1](i-n) + T_N$. This step yields

$$\tau[1](12) = 83 + 96 = 179, \quad \tau[1](13) = 84 + 96 = 180, \quad \ldots$$

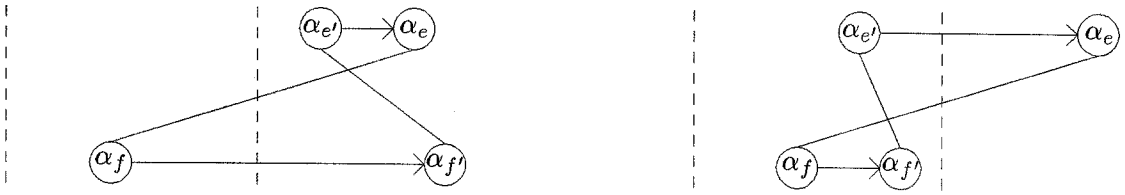


Figure 4.4: Violation of Eqn. (4.17)

Note that the definitions of $\tau[1](13)$ and $\tau[1](17)$ in step 3 and in this step are consistent. It can be shown that this is always the case due to the fact that the sequence function $\sigma[N-2]$ satisfies (4.13) by inductive hypothesis.

By repeatedly applying these procedures, $\{\tau[l](i)\}$ can be constructed for any N. After the construction shown in Figure 4.3, define $\sigma[0]$ and $\sigma[1]$ by (4.14) as

$$\sigma[0](\alpha_0,t) = 48+96t; \qquad \sigma[1](\alpha_0,t) = 83+96t;$$

$$\sigma[0](\alpha_1,t) = \begin{cases} 0 & \text{if } t=0 \\ 60+96(t-1) & \text{if } t \geq 1; \end{cases} \qquad \sigma[1](\alpha_1,t) = \begin{cases} 36 & \text{if } t=0 \\ 84+96(t-1) & \text{if } \geq 1; \end{cases}$$

$$\cdots$$

$$\sigma[0](\alpha_6,t) = \begin{cases} 12 & \text{if } t=0 \\ 120+96(t-1) & \text{if } t \geq 1; \end{cases} \qquad \sigma[1](\alpha_6,t) = \begin{cases} 48 & \text{if } t=0 \\ 144+96(t-1) & \text{if } t \geq 1; \end{cases}$$

$$\cdots$$

Then $\{\sigma[0],\sigma[1]\}$ constitutes a CSSF for $\mathcal{P}$ and $N=2$; for instance, $\sigma[0](\alpha_0,t) = \sigma[1](\alpha_6,t)$ as required by Def. 2.5. □

There are several consequences of Theorem 3 that will be useful later. To describe them, the definition below is needed.

Definition 4.8 *For a canonical program $\mathcal{P}$, the* **set of maximum correspondences** *is*

$$\mathrm{C}_{\max} = \{(\alpha_e,\alpha_f) \mid (\alpha_e,\alpha_f) \in \mathrm{C} \wedge \Delta(\alpha_e,\alpha_f) = 1\},$$

and the **set of minimum correspondences** *is*

$$\mathrm{C}_{\min} = \{(\alpha_e,\alpha_f) \mid (\alpha_e,\alpha_f) \in \mathrm{C} \wedge \Delta(\alpha_e,\alpha_f) = 0\}.$$

Now, for a deadlock-free program, $\bar{\mathrm{C}}$ satisfies the monotonicity condition, so it follows that $\mathrm{C}_{\max}$ and $\mathrm{C}_{\min}$ satisfy the condition as well. Moreover, consider the situation depicted in Figure 4.4, where the correspondence (α_e,α_f) has a phase difference of 1, and $(\alpha_{e'},\alpha_{f'})$ has a phase difference of 0. On the left side of the figure, $e' < e$ and a cycle of dependencies is developed; on the right side, $f' > f$ and, again, deadlock ensues. These observations lead to the following lemma:

Lemma 4.3 *If $\mathcal{P}$ is a deadlock-free canonical program, then*

$$\forall\, e,f,e',f' : (\alpha_e,\alpha_f),(\alpha_{e'},\alpha_{f'}) \in \mathrm{C}_{\max} : e < e' \Leftrightarrow f < f', \tag{4.15}$$

$$\forall\, e,f,e',f' : (\alpha_e,\alpha_f),(\alpha_{e'},\alpha_{f'}) \in \mathrm{C}_{\min} : e < e' \Leftrightarrow f < f', \tag{4.16}$$

$$\forall\, e,f,e',f' : (\alpha_e,\alpha_f) \in \mathrm{C}_{\max} \wedge (\alpha_{e'},\alpha_{f'}) \in \mathrm{C}_{\min} : e < e' \wedge f > f'. \tag{4.17}$$

4.5 Cycles

Let $\mathcal{P}$ be a canonical program and $\bar{\mathcal{P}} = \langle \bar{\mathrm{A}}, \bar{\Gamma}, \bar{\mathrm{C}} \rangle$ be its m-step expansion for some integer m. It should not be too difficult to see that if $\mathcal{P}$ has CRT, then $\bar{\mathcal{P}}$ must satisfy the uni-direction condition, since otherwise there would be two occurrences in $p[0]$ that differ in sequence steps by at least $2N-2$ (recall Figure 3.3). For convenience, we make the following definition.

Definition 4.9 *For any positive integer m, let $\bar{\mathcal{P}} = \langle \bar{\mathrm{A}}, \bar{\Gamma}, \bar{\mathrm{C}} \rangle$ be the m-step expansion of a program $\mathcal{P}$. Then, a correspondence (β_e, β_f) of $\bar{\mathrm{C}}$ is said to be* **pointing left** *if $f < e$.*

For a canonical program $\mathcal{P}$, there exists a correspondence (α_g, α_h) in C with a phase difference of one; this phase difference induces a left-pointing correspondence in $\bar{\mathrm{C}}$. Thus, all other correspondences in any m-step expansion of $\mathcal{P}$ must also be pointing left if $\mathcal{P}$ is to have CRT. However, the example below will serve to show that the monotonicity and uni-direction conditions are not sufficient to imply that a pseudo-cyclic program has CRT.

Example 4.9: Let $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ be a canonical program with

$$\begin{aligned} \mathrm{A} &= \{\alpha_0, \alpha_1, \ldots, \alpha_7\}; \\ \Gamma &= \alpha_5\alpha_6(\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7)^\star; \\ \mathrm{C} &= \{(\alpha_0, \alpha_5), (\alpha_3, \alpha_6), (\alpha_4, \alpha_1), (\alpha_7, \alpha_2)\}. \end{aligned}$$

Thus, the set of maximum correspondences is $\mathrm{C}_{\max} = \{(\alpha_0, \alpha_5), (\alpha_3, \alpha_6)\}$, and the set of minimum correspondences is $\mathrm{C}_{\min} = \{(\alpha_4, \alpha_1), (\alpha_7, \alpha_2)\}$.

Note that any m-step expansion of $\mathcal{P}$ satisfies the uni-direction and monotonicity conditions; for instance, Figure 4.5a shows the correspondence graph for the two-phase expansion of $\mathcal{P}$. However, $\mathcal{P}$ does not have CRT, as we shall now show.

Consider Figure 4.5b which shows the correspondences between adjacent processes implementing $\mathcal{P}$. The part of the communication pattern shown in the figure is assumed to be well into the steady-state region (*i.e.*, t is large). Notice that by the phase difference of 1 in the correspondence (α_3, α_6), the occurrence of α_3 in phase t of process $p[l]$ coincides with the occurrence α_6 in phase $t-1$ of process $p[l+1]$. Also, the latter occurrence must precede α_7 in phase $t-1$ of $p[l+1]$ which, in turn, coincides with α_2 in phase $t-1$ of $p[l+2]$. Thus, α_3 in phase t of $p[l]$ precedes α_3 in phase $t-1$ of $p[l+2]$ by at least two steps. So, for N processes, where N is odd, α_3 in phase t of $p[0]$ must precede α_3 in phase $t-(N-1)/2$ of $p[N-1]$ by at least $N-1$ steps. Similarly, α_4 in phase t of $p[0]$ must *follow* α_4 in phase $t-(N-1)/2$ of $p[N-1]$ by at least $N-1$ steps. Hence, no CRT is possible, since α_3 must precede α_4 in phase t of $p[0]$ by at least $2N-2$ steps. □

Figure 4.6 illustrates the general case. Let $L > 0$ and D be two integers. Suppose that there is an action α_i such that its occurrence in phase t of $p[l]$ must *precede* its occurrence in phase $t-D$ of $p[l+L]$ by at least one step (as indicated by the southwest precedence arrow). Suppose also that there is another action α_j with the reversed property that its occurrence in phase t of $p[l]$ must *follow* its occurrence in phase $t-D$ of $p[l+L]$. Then the corresponding program cannot have CRT, as indicated by the fact that as N increases, the number of steps separating α_i and α_j in $p[0]$ must also increase.

Example 4.10: As the argument in the previous example shows, there do exist $L = 2$, $D = 1$, $\alpha_i = \alpha_3$, and $\alpha_j = \alpha_4$ which satisfy the discussion above.

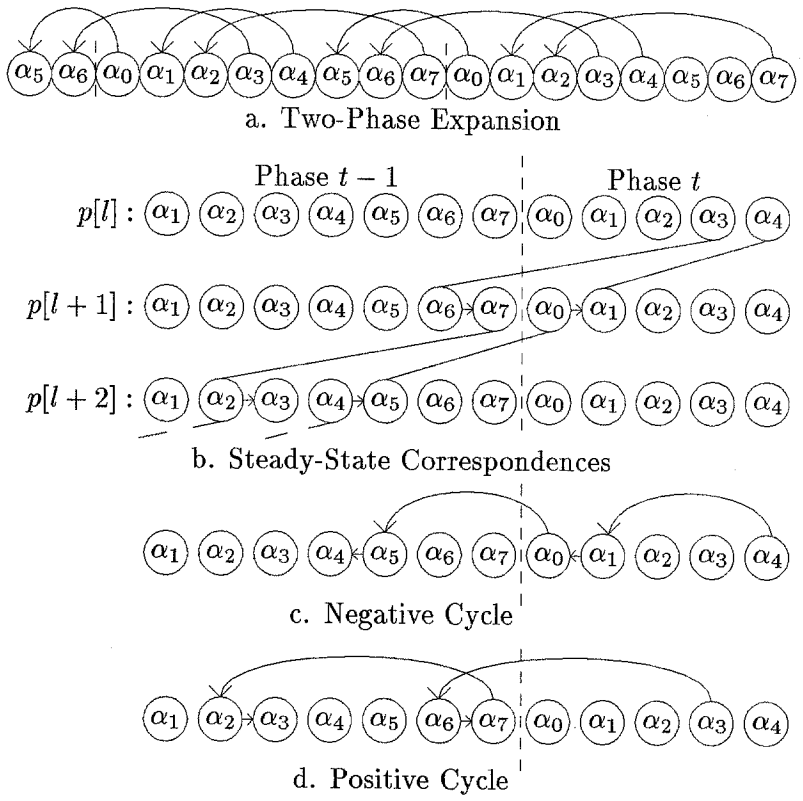


Figure 4.5: Presence of Negative and Positive Cycles in Example 4.9

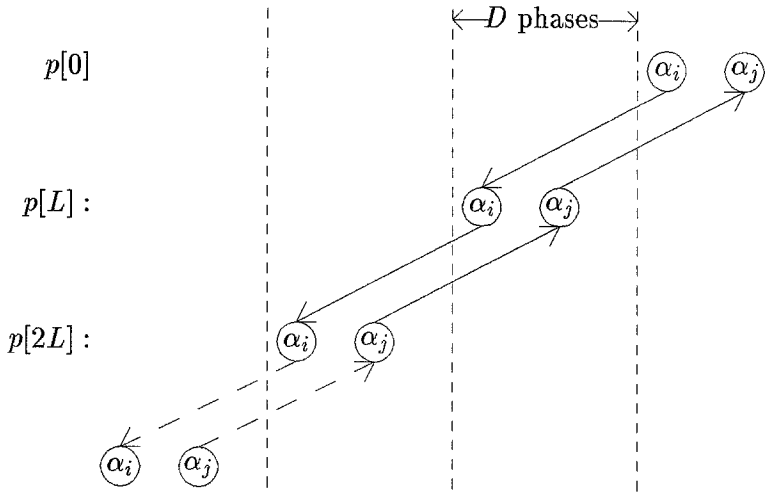


Figure 4.6: No Constant Response Time

We will now investigate how these values are related to the program specification. In Figure 4.5c, we have indicated the correspondences that have resulted in the precedence relationship for α_4. They are represented as a path[2] in the correspondence graph of some m-step expansion of $\mathcal{P}$. There are several notable features which are listed below; their significance will be discussed immediately after this example.

- The beginning and the end of the path correspond to the same action, namely, α_4; thus, we call the path a "cycle".
- All non-correspondence edges (the horizontal ones) are pointing to the left; therefore it is said to be "negative".
- The path contains two correspondence edges (the curved arcs); therefore, it is said to have "length" 2.
- The path spans exactly one phase; therefore, it is said to be of "period" 1.

Similarly, Figure 4.5d shows the correspondences for the precedence relationship for α_3. This path is called "a positive cycle of length 2 and period 1"; it is positive because all horizontal edges are pointing to the right. □

Figure 4.7a and Figure 4.7b illustrate how the existence of a negative and a positive cycle of length L and period D induces the precedence relations of Figure 4.6. Notice that the condition for a cycle to be "negative" or "positive" is necessary. If both left- and right-pointing horizontal edges exist, then no conclusion can be made about the ordering of the occurrences as shown in Figure 4.7c.

The following definitions formalize these concepts. The set C° contains the correspondences in one phase in the steady-state region of the communication pattern. Note that due to the phase difference of 1 inherent in a correspondence $(\alpha_e, \alpha_f) \in C_{\max}$, the actual correspondence is between α_e of the current phase and α_f of the *previous* phase; therefore, the correspondence $(e, f-n)$ is included in C°. The set $C^\star$ attempts to capture the steady-state behavior of the program by specifying the correspondences for an unbounded number of phase[3] in the body of Γ.

Definition 4.10 *For a canonical program $\mathcal{P}$, the* **set of steady-state correspondences** *is*

$$C^\circ = \{(e, f - n\Delta(\alpha_e, \alpha_f)) \mid \exists\, e, f :: (\alpha_e, \alpha_f) \in C\}$$

and the **extended set of steady-state correspondences** *is*

$$C^\star = \{(e, f) \mid \exists\, j, \hat{e}, \hat{f} : j \in \mathbf{Z} \wedge (\hat{e}, \hat{f}) \in C^\circ : e = \hat{e} + nj \wedge f = \hat{f} + nj\}.$$

The definitions of **E()**, **F()**, **ecorr()**, and **fcorr()** are extended in an obvious manner when the correspondences are represented by pairs of integers as in the cases of C° and $C^\star$.

Definition 4.11 *For a program $\mathcal{P}$ with extended set of steady-state correspondences $C^\star$, a* **path** *ρ is a list*

$$\langle (e_0, f_0), (e_1, f_1), \ldots, (e_L, f_L) \rangle$$

[2]To facilitate the discussion, the path is shown with additional horizontal edges that are not actual edges in the correspondence graph.

[3]In all analyses only a predeterminable finite portion of this expansion will be used.

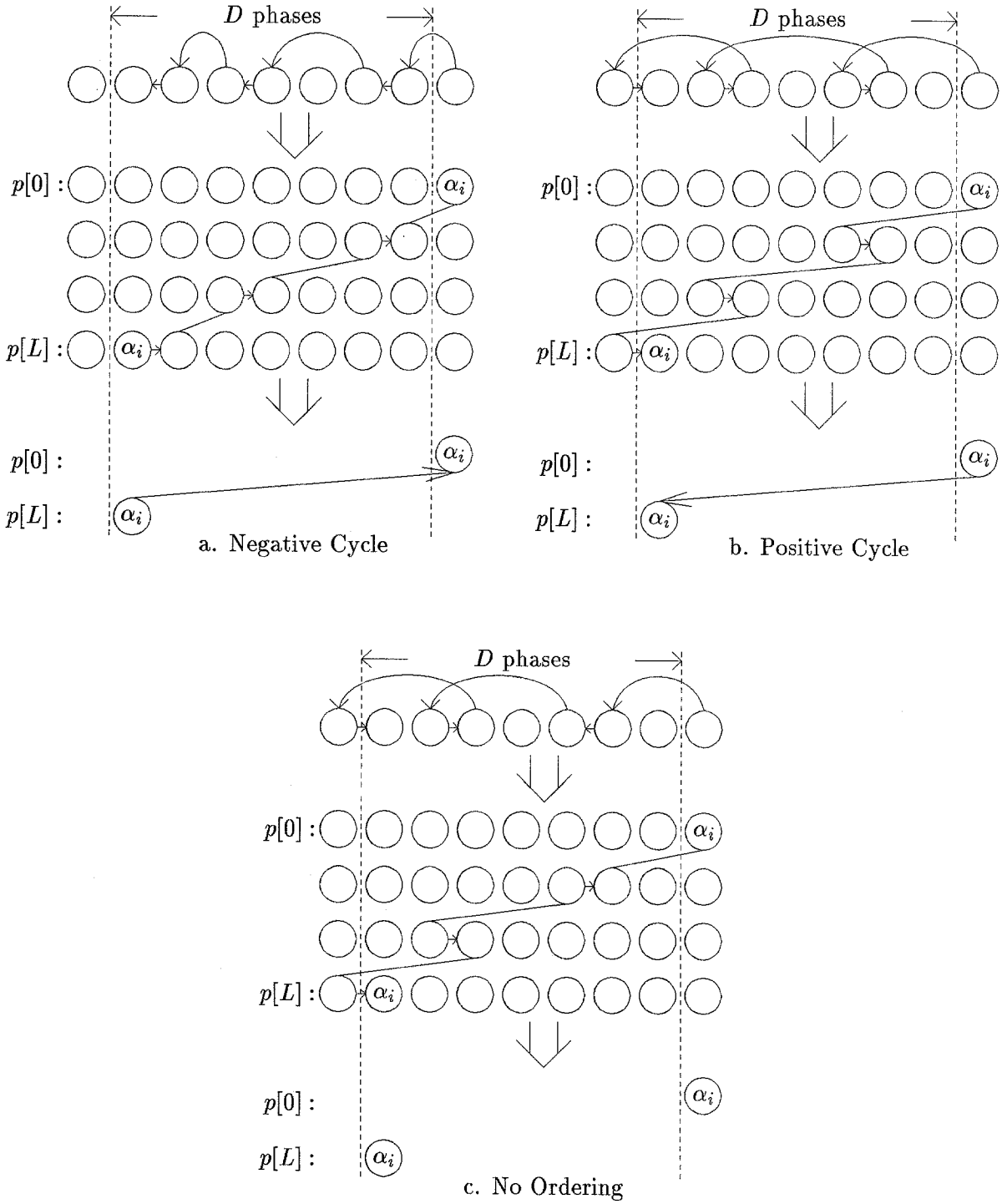


Figure 4.7: Cycles of C*

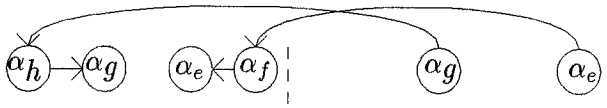


Figure 4.8: Violation of Conditions in Lemma 4.4

such that $L > 0$ *and* $\forall\, i : 0 \leq i \leq L : (e_i, f_i) \in \mathrm{C}^*$. L *is called the* **length** *of the path.*

If $(\forall\, i : 0 \leq i < L : e_{i+1} < f_i)$, *then the path is called a* **negative path**. *If* $(\forall\, i : 0 \leq i < L : e_{i+1} > f_i)$, *then the path is called a* **positive path**.

If $e_0 = e_L + nD$, *then the path is called a* **cycle** *with* **period** D. *Terms like* **positive cycle** *and* **negative cycle** *are defined in the obvious manner.*

4.6 Constant Response Time

Theorem 4 *A canonical program* $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ *has CRT if and only if it satisfies the following:*

Its two-phase expansion satisfies the monotonicity condition. (4.18)

All of the correspondences in the head of Γ *are pointing left.* (4.19)

There does not exist a positive cycle and a negative cycle of C^* *with the same length and period.* (4.20)

We have already argued the necessity of these three conditions. We shall now proceed to show that these three conditions are also sufficient; however, such an accomplishment would not be too practical, since the cycle condition (4.20) must be shown to hold for all possible lengths and periods. Fortunately, we have developed a polynomial-time algorithm that would decide if a canonical program $\mathcal{P}$ satisfies (4.20).

Before presenting the algorithm, some preliminary discussions are needed. First, the following lemma shows that deciding if a program with $h < g$ has CRT is very easy[4]:

Lemma 4.4 *If, for a canonical program* $\mathcal{P}$, $h < g$, *and* $\mathcal{P}$ *satsifies (4.18) and (4.19), then* $\mathcal{P}$ *has CRT if and only if*

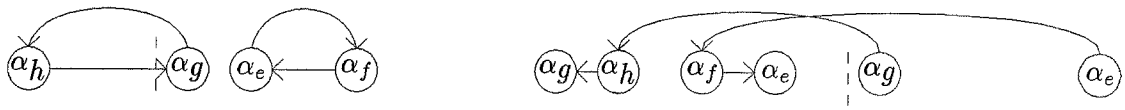
$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C} : (\Delta(\alpha_e, \alpha_f) = 1 \wedge f < e). \tag{4.21}$$

The necessity of (4.21) is relatively easy to show. If there exists a correspondence (α_e, α_f) that violates the condition above, then $\mathcal{P}$ will have positive and negative cycles of period 1 and length 1 as indicated in Figure 4.8. [5] The converse of the claim can also be proven by constructing a CSSF for $\mathcal{P}$ using a modified version of Algorithm 1 and Algorithm 4 (to be described later).

Example 4.11: Going back to the substring recognizer, since there exists a correspondence (α_3, α_0) with phase difference of 0, (4.21) is violated and we can conclude immediately that it does not have CRT. □

[4] Recall the definition of g in (4.1)

[5] The vertical dashed line separates consecutive phases.



a. Violation of Eqn. (4.23)

b. Violation of Eqn. (4.24)

Figure 4.9: Violation of Conditions in Lemma 4.5

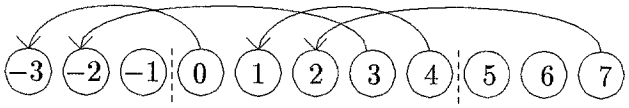


Figure 4.10: Steady-State Expansion Graph of Example 4.9

In the sequel, we shall restrict our analysis to canonical programs satisfying

$$h > g. \tag{4.22}$$

Next, the following lemma can be applied to sieve out another significant portion of programs that violated (4.20).

Lemma 4.5 *If a canonical program $\mathcal{P}$ satisfies (4.20), then*

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C_{min}} : e > f, \tag{4.23}$$
$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C_{max}} : e < f. \tag{4.24}$$

To establish the lemma, by (4.22), $(\alpha_g, \alpha_h) \in \mathrm{C_{max}}$. Figure 4.9a shows the presence of negative and positive cycles of length 1 and period 0 if (4.23) is violated. Figure 4.9b shows the presence of negative and positive cycles of length 1 and period 1 if (4.24) is violated.

Next, we define the **steady-state expansion graph** as the correspondence graph of C°; in other words, it is one phase in the steady-state expansion of Γ, with all those correspondences to occurrences outside the phase included. Figure 4.10 shows the steady-state expansion graph for the program in Example 4.9. Notice the numbering of the vertices: the vertices belonging to the phase are numbered by the indices of the corresponding symbols; those outside the phase are indexed by $-1, -2, \ldots, h - n$. Also, two vertical dashed lines are placed in Figure 4.10b to serve as a reminder that the vertices on the right of both lines represent the same actions as those on the left of both lines; *i.e.*, vertices $-i$ and $n - i$ represent two occurrences of the same action that are one phase apart. For ease of reference, this graph is denoted $\mathrm{G}(\mathrm{C}^\circ)$.

The set of steady-state correspondences of a canonical program for which (4.18), (4.22), (4.23), and (4.24) hold satisfies the following properties:

$$\forall\, e : e \in \mathbf{E}(\mathrm{C}^\circ) : 0 \le e < n, \tag{4.25}$$
$$\forall\, f : f \in \mathbf{F}(\mathrm{C}^\circ) : h - n \le f < h, \tag{4.26}$$
$$\forall\, e, f : (e, f) \in \mathrm{C}^\circ : f < e. \tag{4.27}$$

The first is by the definition of C°. To establish the second, note that Lemma 4.3 and $(g, h) \in \mathrm{C_{max}}$ imply that

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C_{min}} : f < h, \qquad \text{[by (4.1) \& (4.17)]}$$
$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C_{max}} : f - n \ge h - n. \qquad \text{[by (4.16) \& } e \ge 0\text{]}$$

Also, for $(\alpha_e, \alpha_f) \in \mathrm{C}^\circ$, if $(\alpha_e, \alpha_f) \in \mathrm{C_{min}}$, then $f < e$ by (4.23); alternatively, if $(\alpha_e, \alpha_{f+n}) \in \mathrm{C_{max}}$, then, by $\alpha_{f+n} \in \mathrm{A}$, $f + n < n \leq e + n$. Thus, in either case, any correspondence in C° must be pointing left.

From these observations, we make the the following definition[6].

Definition 4.12 *Let* $(e, f) \in \mathrm{C}^\star$. *The* **projection** *of* (e, f), *denoted by* **proj**$((e, f))$, *is the correspondence* $(\varepsilon(e), \phi(f)) \in \mathrm{C}^\circ$ *where*

$$\varepsilon(e) \stackrel{\text{def}}{=} e \bmod n, \tag{4.28}$$

$$\phi(f) \stackrel{\text{def}}{=} ((f - h) \bmod n) - n + h, \tag{4.29}$$

Note that if $(e, f) \in \mathrm{C}^\star$, then there exist $\varepsilon(e)$, $\phi(f)$, and j, such that

$$e = \varepsilon(e) + nj \wedge f = \phi(f) + nj \wedge (\varepsilon(e), \phi(f)) \in \mathrm{C}^\circ;$$

thus, by (4.25) and (4.26), Definition 4.12 is consistent.

We are now ready to present Algorithm 2 which decides if the cycle condition holds for a given program. Upon exit from the algorithm, the variable *status* will be set to cycles_found if a positive and a negative cycles of the same length and period are found; otherwise, it will be set to no_cycles_found. As we shall see, in the latter case, a consistent set of sequence functions can then be constructed for $\mathcal{P}$ provided (4.19) is also valid.

The algebraic specification of the algorithm and its associated Procedure 2A is in the Appendix; below is their description in a more intuitive form. Additional remarks follow afterward.

ALGORITHM 2:
Input: A pseudo-cyclic canonical program $\mathcal{P}$ satisfying (4.18), (4.22), (4.23), and (4.24).
Output: The variable *status* which is set to cycles_found if $\mathcal{P}$ does not satisfy (4.20) and is set to no_cycles_found otherwise.

1. Set k to be $h - n$.
2. If $k = n - 1$, then set *status* to no_cycles_found and exit algorithm; else increment k.
3. If vertex k is not a tail vertex in the steady-state expansion graph then go back to step 2; else, perform the remaining steps.
4. Let B contains only the edges not to the right of vertex k in the steady-state expansion graph. Set w to k and then execute Procedure 2A. If the procedure sets *status* to cycles_found, then exit.
5. Next, reverse the sense of ordering on the vertices by the mapping

   $$\psi(i) = h - 1 - i.$$

   Also, reverse the direction of the correspondences in B. Set w to $\psi(k - n - 1)$. Again, apply Procedure 2A and exit if *status* is set to cycles_found.

[6]The infix operator **mod** is defined by

$$j = i \bmod n \Leftrightarrow (j \equiv i \pmod n \wedge 0 \leq j < n).$$

6. Go to Step 2.

PROCEDURE 2A:

1. Define $B^\star$ to be $\{(e,f) \mid \exists\, e,f,j : (\hat{e},\hat{f}) \in B \wedge j \in \mathbf{Z} : e = \hat{e} + nj \wedge f = \hat{f} + nj\}$. Also, let G(B) and G($B^\star$) denote the correspondence graphs of B and $B^\star$, respectively.

2. Set the edge $(ey[0], fy[0])$ to be the edge immediately to the left of w in G(B) ($ey[0] := \max\{i \mid i < w \wedge i \in \mathbf{E}(B)\}$). Set the edge $(ex[0], fx[0])$ to be the leftmost edge not to the right of vertex w ($ex[0] := \min\{i \mid i \geq w \wedge i \in \mathbf{E}(B)\}$). Also, initialize ρ_x to contain only $(ex[0], fx[0])$ and ρ_y to contain only $(ey[0], fy[0])$. Set κ to 0.

3. Check if $fx[\kappa] < ey[\kappa]$. If not, set *status* to **no_cycles_found** and exit.

4. Increment κ.

5. Extend ρ_x by adding the leftmost edge in G($B^\star$) that would still maintain ρ_x as a positive path. Name that edge $(ex[\kappa], fx[\kappa])$. Similarly, extend ρ_y by adding the rightmost edge, $(ey[\kappa], fy[\kappa])$, that would still maintain it as a negative path.

6. Check if ρ_x and ρ_y contain cycles of same length and period. If so, set *status* to **cycles_found** and exit procedure; else, go to Step 3.

We'll next illustrate the operations performed by this Algorithm. The argument for its correctness will follow.

Example 4.12: Let $\mathcal{P} = \langle A, \Gamma, C\rangle$ be a canonical program with

$$\begin{aligned} A &= \{\alpha_0, \alpha_1, \ldots, \alpha_9\}; \\ \Gamma &= \alpha_5\alpha_6\alpha_9(\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7\alpha_8\alpha_9)^\star; \\ C &= \{(\alpha_0,\alpha_5),(\alpha_2,\alpha_6),(\alpha_3,\alpha_9),(\alpha_7,\alpha_1),(\alpha_8,\alpha_4)\}. \end{aligned}$$

The first time that Step 4 of Algorithm 2 is executed, $k = 7$. Figure 4.11a shows G(B), the correspondence graph of B constructed in that step, and Figure 4.11b shows G($B^\star$). The paths ρ_x and ρ_y traced out by Procedure 2A for this B are indicated in Figure 4.11c where the correspondence edges for ρ_x are placed above the nodes and those for ρ_y are placed below. Since $fx[2] = -9 \geq -10 = ex[2]$, Procedure 2A exits with *status* = **no_cycles_found.**

With k still being 7, Step 5 of Algorithm 2 is performed and a new set of correspondences is assigned to B as depicted in Figure 4.11d. Note that this new set is simply a mirror image of the one constructed in Step 4 with the directions of the correspondences reversed. Next w is set to $\psi(k-n-1) = 8$ and two paths, as indicated in Figure 4.11e, are traced out in G($B^\star$). Again, no cycles are found.

For the next execution of Step 4 with $k = 8$, G(B) and the two paths constructed by Procedure 2A are shown in Figure 4.11f and Figure 4.11g, respectively. Once again, no cycles are found.

Next, Step 5 of Algorithm 2 is performed with $k = 8$. The set B constructed by this step is shown in Figure 4.11h. The variable w is set to $\psi(k-n-1) = 7$. Now, as indicated in Figure 4.11i, for $\kappa = 2$, ρ_x contains the positive cycle

$$\langle (8,2),(3,-3),(-2,-8)\rangle,$$

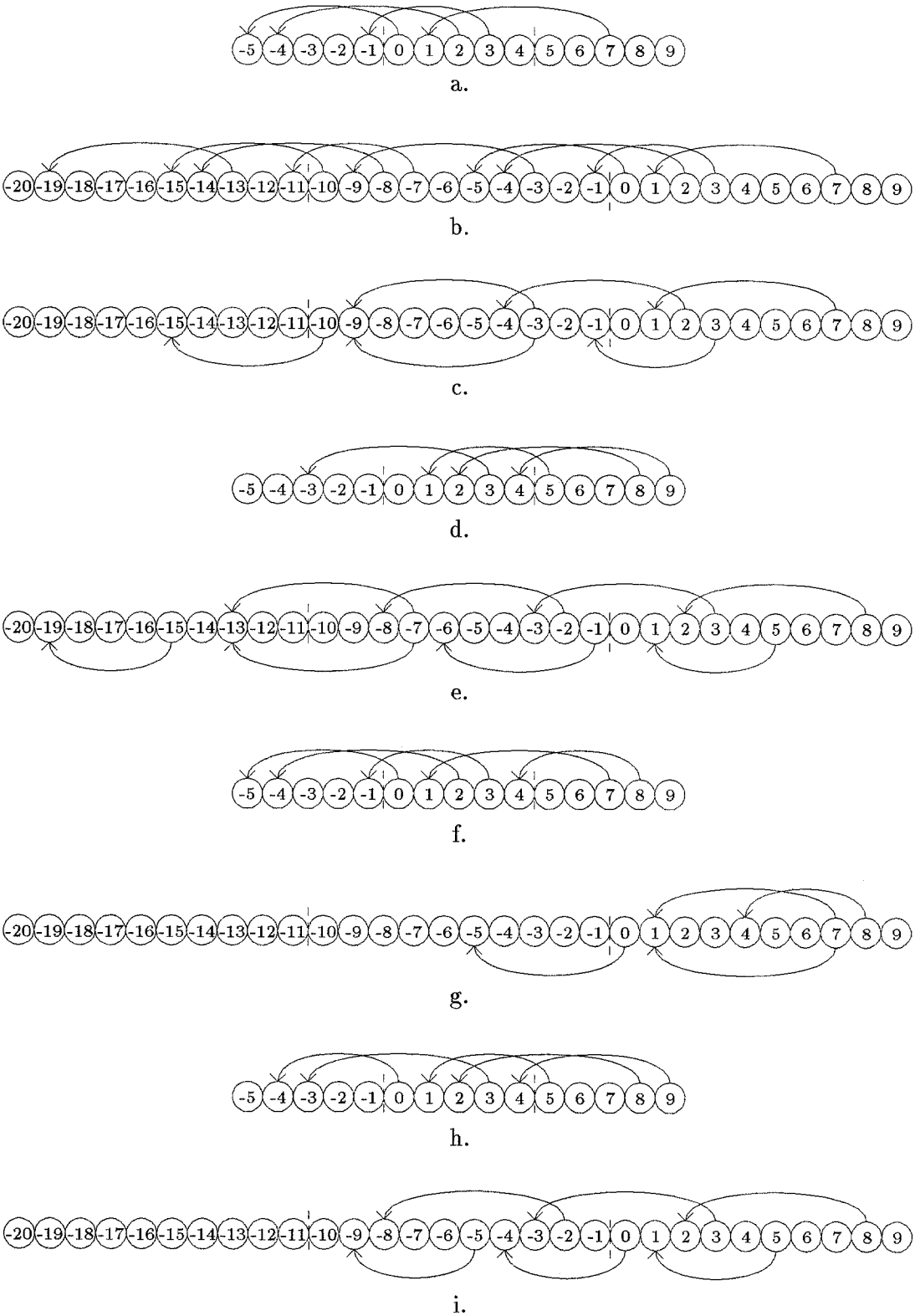


Figure 4.11: Application of Algorithm 2 for Example 4.12

and ρ_y contains the negative cycle

$$\langle (5,1),(0,-4),(-5,-9)\rangle.$$

Since the two cycles have the same length 2 and the same period 1, Procedure 2A exits after setting *status* to cycles_found. It should not be difficult to see that the two cycles found by the algorithm induce corresponding cycles in $C^\star$. Thus, $\mathcal{P}$ does not satisfy the cycle condition and the algorithm behaves correctly for this example. □

Now we present some argument for the correctness of Algorithm 2. First, we claim that the selections in Step 2 of Procedure 2A can always be made. Note that when the procedure is invoked, there must exist $f > k-n$ such that $(k,f) \in C^\circ$ since k is a tail vertex and no correspondence spans more than n vertices. Now, by definition, $(g, h-n) \in C^\circ$. So when Procedure 2A is called in Step 4 of Algorithm 2, we have $g < w \leq k$ with both g and k in B. When it's called in Step 5, since $f > k-n-1 \geq h-n$, $\psi(f) < w \leq \psi(h-n)$ with both $\psi(f)$ and $\psi(h-n)$ in B.

To show that Procedure 2A terminates, let $\rho_x = \langle \xi_x[0], \xi_x[1], \ldots, \xi_x[\kappa-1]\rangle$ and $\rho_y = \langle \xi_y[0], \xi_y[1], \ldots, \xi_y[\kappa-1]\rangle$ be preconditions of Step 5 in the procedure. Let $\xi_x[\kappa]$ and $\xi_y[\kappa]$ be the two edges selected by that step. Suppose that $\xi_x[\kappa]$ has the same projection as $\xi_x[i]$ for an i between 0 and $\kappa-1$. Then, $\langle \xi_x[i], \xi_x[i+1], \ldots, \xi_x[\kappa]\rangle$ is a positive cycle of length $\kappa - i$. Similarly, if $\xi_y[\kappa]$ has the same projection as $\xi_y[i]$, then there is a negative cycle of length $\kappa - i$. Furthermore, these two cycles must have the same period, since if one path is more than one cycle ahead of the other, then the exit condition in Step 3 would have been satisfied previously. Therefore, Step 5 of Procedure 2A can be implemented by maintaining the set $S = \{\langle \mathbf{proj}(\xi_x[i]), \mathbf{proj}(\xi_y[i])\rangle \mid 0 \leq i < \kappa\}$ and checking if $\langle \mathbf{proj}(\xi_x[\kappa]), \mathbf{proj}(\xi_y[\kappa])\rangle$ is in the set. This is described explicitly in the Appendix. Since, there can be at most $|C|^2$ distinct entries in S, the algorithm will terminate with a worst-case time complexity of $O(|C|^3)$.

Next, it should be fairly obvious that, when called from Step 4 of Algorithm 2, Procedure 2A sets *status* to cycles_found implies that the input program contains positive and negative cycles of same length and period since B is a subset of C. Moreover, for a given k, let B_1 be the value of B as constructed in Step 4 of Algorithm 2 and B_2 be its value in Step 5. It is not difficult to see that there's an isomorphic mapping between positive (negative) cycles in $B_1^\star$ and positive (negative) cycles in $B_2^\star$. Thus, if the algorithm sets *status* to cycles_found, then the input program does not satisfy the cycle condition. However, it is not at all obvious that the following claim is true:

Lemma 4.6 *Let $\mathcal{P}$ be a canonical program that satisfies (4.18), (4.19), (4.22), (4.23), and (4.24). Then, $\mathcal{P}$ has CRT if and only if, upon the termination of Algorithm 2 with $\mathcal{P}$ as input, the variable status is set to* no_cycles_found.

As for the cyclic case, we will prove this statement by explicitly constructing a consistent set of sequence functions that induces CRT. Our method is to apply Algorithm 3 to assign to each of the $2n-h$ vertices in the steady-state expansion graph an "η-value" so that the difference between the values assigned to the tail and head of each edge (*i.e.*, the "η-difference" of the edge) is a constant M. However, we also need to produce a constant T for the length of each phase, and to maintain that vertex $-i$ is precisely one phase ahead of vertex $n-i$, since these two vertices represent different occurrences of the same action.

In algebraic terms, we wish to produce T, M, and $\{\eta(i) \mid h-n \leq i < n\}$ with the following properties:

$$\forall\, i : h-n \leq i < n-1 : \eta(i) < \eta(i+1), \tag{4.30}$$

$$\forall\, e,f : (e,f) \in \mathrm{C}^\circ : \eta(e) = \eta(f) + M, \tag{4.31}$$

$$\forall\, i : h \leq i < n : \eta(i) = \eta(i-n) + T. \tag{4.32}$$

Once these values have been obtained, we then find appropriate sequence steps for the head of Γ and a CSSF for $\mathcal{P}$ can be constructed.

Similar to the cyclic case, our approach is to assign $\eta(k)$ as k ranges from $h-n$ to $n-1$ while maintaining the following invariants:

$$\forall\, e,f : (e,f) \in \mathrm{C}^\circ \wedge f \leq k : \eta(e) = \eta(f) + M, \tag{4.33}$$

$$\forall\, i : h \leq i \leq k : \eta(i) - \eta(i-n) = T. \tag{4.34}$$

The following example will show that satisfying both invariants is not trivial.

Example 4.13: Let $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ be a canonical program with

$$\begin{aligned} \mathrm{A} &= \{\alpha_0, \alpha_1, \ldots, \alpha_7\}; \\ \Gamma &= \alpha_4\alpha_6(\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7)^\star; \\ \mathrm{C} &= \{(\alpha_0,\alpha_4),(\alpha_3,\alpha_6),(\alpha_5,\alpha_1),(\alpha_7,\alpha_2)\}. \end{aligned}$$

If we apply Algorithm 1 of Chapter 3 to the leftmost nine vertices of the steady-state expansion graph of $\mathcal{P}$, the result is shown in Figure 4.12a — the η-difference for the edges is $M = 5$ at this stage. Note also that T, the length of the period is currently set at $\eta(4)-\eta(4-n) = 9-0 = 9$. Next, $\eta(5)$ is set to $\eta(-3)+T = 11$ and $\eta(6)$ to $\eta(-2)+T = 12$. For these assignments, no updating of $\{\eta(i)\}$ is needed to maintain the invariants. The assignments are shown in Figure 4.12b.

Now, for vertex 7, we first assign it a η-value of $\eta(7) = \eta(-1) + T = 13$. Since the η-difference for the edge containing vertex 7 is $\eta(7) - \eta(2) = 6 > M$, we need to modify the $\eta(i)$'s so as to maintain (4.33). Suppose we try using Procedure 1A to stretch all the edges to the left of vertex 7 by $\delta = \eta(7) - \eta(2) - M = 1$. Figure 4.12b show where each δ would be placed — the index of each δ indicates the order of placement. However, note that we are not done. By placing a δ between vertices -4 and -3, we have caused the value of $\eta(4) - \eta(4-8)$ to be δ more than that of $\eta(i) - \eta(i-n)$ for $i = 5, 6, 7$, thereby violating (4.34). (Recall that η-values to the right of m δ's are incremented by $m\delta$.) One obvious solution is to place an additional δ between vertices 4 and 5, as indicated by δ_2 in Figure 4.12c, to compensate for the difference. We call this technique — where the placement of a δ between vertices i and $i+1$ is followed by the placement of another δ between vertices $n+i$ and $n+i+1$ to maintain the constant phase difference — a "wrap-around" since the leftmost h vertices and the rightmost h vertices represent the same actions occurring one phase apart.

Now, edges (5,1) and (7,2) are stretched again by the wrap-around; so we continue with Procedure 1A and find the rightmost edge that needs to be stretched, *i.e.*, (3,−2). Following the procedure, a δ is placed to the right of vertex -2. Again, a wrap-around is needed and so we put an additional δ between vertices 6 and 7. But doing so would result in

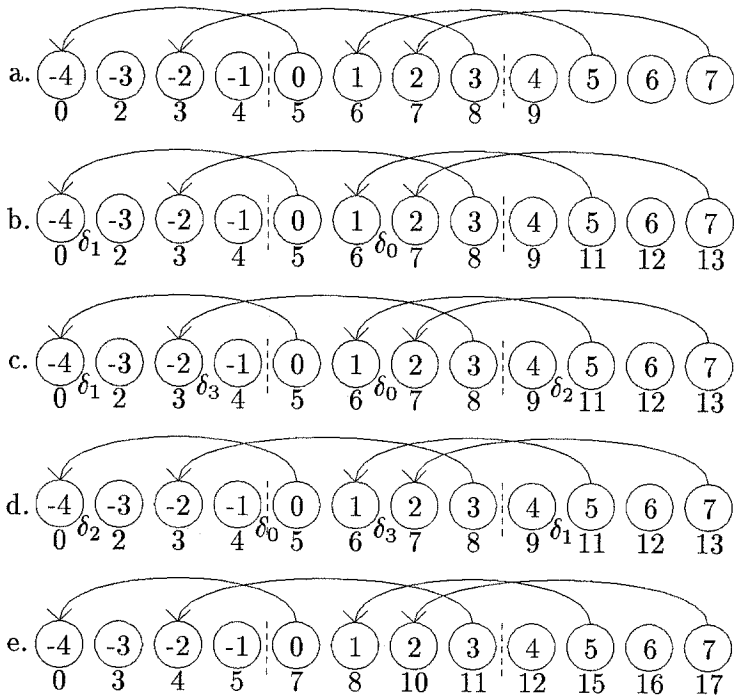


Figure 4.12: Construction of $\{\eta(i)\}$ for Example 4.13

stretching the edge (7,2) and we would end up where we began, namely, needing to increase the η-differences of all edges to the left of vertex 7 by δ. However, this failed attempt does serve a purpose, the significance of which will become clear later. We now know that we can stretch the edge (3,−2) and all edges[7] to the right of it by placing a δ between vertices 1 and 2.

Right now, we need some other method to stretch the edges so as to maintain the invariants. Suppose that we place a δ to the right of vertex -1 and work "backward". We next find the leftmost unstretched edge and place a δ immediately to the left of its tail. At this point, a wrap-around is necessary to preserve the constant phase length. The δ placements performed thus far are indicated by δ_0, δ_1, and δ_2 of Figure 4.12d.

After the wrap-around, the edge (0,−4) has been stretched twice; so, to compensate, all edges[8] to the right of it must be stretched. But these are exactly the edges that would be stretched if we place a δ between vertices 1 and 2 as shown earlier by our unsuccessful attempt to apply Procedure 1A. This placement is indicated by δ_3 in Figure 4.12d. and the final $\eta(i)$ assignments are shown in Figure 4.12e. □

The previous example outlines the method for increasing the η-differences for all the edges to the left of a given vertex w. We start by placing a δ symbol to the right of vertex -1 and work "backward" until we reach a point when we know that a slightly modified version of Procedure 1A can be applied to stretch the remaining edges. These steps are encapsulated in Procedure 3A which, perhaps not too surprisingly, uses the two paths ρ_x and ρ_y constructed by Procedure 2A. Below is then the description of Algorithm 3 and its associated procedures.

[7]That is, all edges except (7,2) which we do not want to stretch.

[8]Again, except the edge (7,2).

ALGORITHM 3:
Input: A pseudo-cycle canonical program $\mathcal{P}$ satisfying (4.18), (4.22), (4.23), and (4.24).
Output: The variable *status* is set to **cycles_found** if $\mathcal{P}$ does not satisfy the cycle condition. Else, the algorithm returns integers T, M, and a set of integers $\{\eta(i) \mid h - n \leq i < n\}$ for which (4.30), (4.31), and (4.32) hold.

1. Apply Algorithm 1 to the leftmost $n + 1$ vertices for the initial values of M and $\{\eta(i) \mid g - n \leq i \leq n - h\}$. Initialize T to $\eta(h) - \eta(h - n)$. Set k to h.

2. If $k = n - 1$ or the program violates the cycle condition, exit; else, increment k.

3. Set $\eta(k)$ to $\eta(k - n) + T$ so as to maintain the constant phase length.

4. If k is not a tail vertex then go to Step 2. If there exists edge (k, k'), let d be its η-difference.

5. If $d = M$, then the invariants are maintained; so, go back to Step 2.

6. If $d > M$, then set w to k, δ to $d - M$, v to $h - n$, and u to k. The last two variables specify the range of indices for which $\eta()$ is defined. Also, set B to exclude all edges to the right of vertex k. Then, apply Procedure 3A so that all edges to the *left* of (k, k') have their η-differences increased by δ while maintaining (4.34).

7. If $d < M$, then set δ to $M - d$, and place a δ immediately to the left of vertex k. This will increase the η-difference of edge (k, k') to M but $\eta(k) - \eta(k - n)$ will no longer be T. So, perform a wrap around by placing a δ to the left of vertex $k - n$ and increment T by δ to maintain (4.34). But now all edges to the *right* of vertex $k - n - 1$ must have their η-difference increased by δ in order to maintain (4.33). This is accomplished by swapping the right and left senses on the vertices and reversing the correspondence directions.[9] Now, if w is set to the reversed image of vertex $k - n - 1$ (*i.e.*, $\psi(k - n - 1)$), then applying Procedure 3A would yield the proper δ placement that would maintain both invariants.

PROCEDURE 3A:

1. Execute Procedure 2A to construct ρ_x and ρ_y, positive and negative paths in B*.

2. If *status*= **cycles_found**, then abort algorithm; the program does not have CRT. Else, apply Procedure 3B which corresponds to the "backward" δ placement described earlier. Afterward, apply Procedure 3C to perform the remaining δ placement that would maintain (4.33) and (4.34) when this procedure exits.

3. Set M to $M + M_1 + M_2 + \delta$ where M_1 and M_2 are defined by Procedures 3B and 3C, respectively.

PROCEDURE 3B:

1. Initialize all edges in G(B) as black. Also, v is the leftmost head vertex.

[9] In other words, $v = \psi(k)$, $u = \psi(h - n)$, and $\mathrm{B} = \{(\psi(f), \psi(e)) \mid (e, f) \in \mathrm{C}^\circ \wedge f \leq k\}$.

2. Let $(ez[0], fz[0])$ be an edge in G(B*) which projects into the leftmost edge in G(B); *i.e.*, $fz[0] = v - jn$ for some j. Also, WLOG, assume $fz[0] \leq fx[\kappa]$ by choosing j small enough.

3. Set j and M_1 to 0 and initialize ρ_z to contain only $(ez[0], fz[0])$.

4. Let **proj**$((ez[j], fz[j]))$ be $(\hat{e}, \hat{f})$.

5. If $(\hat{e}, \hat{f})$ is the same projection as one of the edges from ρ_x, the set ξ to that edge and exit. Otherwise, place a δ immediately to the left of vertex $\varepsilon(\hat{e})$ in G(B). Consider all edges in B stretched by this δ as red. Also, increment T by δ.

6. If the δ is placed to the right of vertex $n + v$, then consider all edges as black and perform a wrap-around by placing a δ to the left of $\hat{e} - n$. All edges stretched by the δ for the wrap-around are colored red. Also, increment M_1 by δ.

7. Increment j and let $(ez[j], fz[j])$ be the leftmost edge that can be added to the *front* of ρ_z and still maintain it as a negative path in B*. Go to Step 4.

PROCEDURE 3C:

1. Initially, all edges in G(B) are considered *black*. Also, let u be the rightmost tail vertex.

2. Color the edges not to the left of vertex w red.

3. Initialize i and M_2 to 0.

4. Let $(\hat{e}, \hat{f})$ be the projection of $(ex[i], fx[i])$ onto the correspondence graph of B.

5. If $(ex[i], fx[i])$ is ξ, then exit algorithm. Otherwise, place a δ immediately to the left of vertex $\hat{f}$. Any edge stretched by this δ is colored red. Also, increment T by δ.

6. If the δ is placed to the left of vertex $u - n$, then color all edges black and perform a wrap-around by placing a δ to the left of vertex $\hat{f} + n$. Any edge stretched by the δ for the wrap-around is colored red. Also, increment M_2 by δ.

7. Increment i and go to Step 4.

Since Algorithm 3 is an extended version of Algorithm 2, if Procedure 2A sets *status* to **cycles_found**, then the cycle condition is violated. Now, assuming that does not occur, we will now argue for the correctness of Algorithm 3.

First, we claim that when Procedure 3A is entered,

$$\forall\, e, f : (e, f) \in \mathrm{B} : f < n + v \wedge 0 \leq e. \tag{4.35}$$

By (4.25) and (4.26), if Procedure 3A is called from Step 4 of Algorithm 3, then the claim clearly holds. If the procedure is called from Step 5 of the algorithm, then let $(e', f') \in \mathrm{C}^\circ$ with $e = \psi(f')$ and $f = \psi(e')$. Now,

$$\begin{aligned} & f' \leq h - 1 && \text{[by (4.26)]} \\ \Rightarrow \quad & e \geq \psi(n - h - 1) = 0 && \text{[by def. of } \psi()] \end{aligned}$$

and

$$\begin{aligned} & e' \geq 0 > k - n && \text{[by (4.25) and } n > k\text{]} \\ \Rightarrow \quad & f < n + (h - 1 - k) = n + \psi(k). && \text{[by def. of } \psi()\text{]} \end{aligned}$$

Thus, (4.35) is established.

Next, we claim that $\mathrm{Q0}(\hat{f})$ is a precondition of Step 5 of Procedure 3B where:

$$\mathrm{Q0}(X) \equiv \text{"For all edges } (e,f) \text{ in G(B), } f < X \text{ implies } (e,f) \text{ is red and } f \geq X \text{ implies } (e,f) \text{ is black."}$$

The predicate $\mathrm{Q0}(\hat{f})$ clearly holds when Step 5 is first executed. Assume, as inductive hypothesis, that it holds prior to the i^{th} execution of Step 5 and that the procedure does not exit during this execution. The placement of δ to the left of $\hat{e}$ stretches exactly those edges (e,f) which satisfy

$$\hat{f} \leq f < \hat{e} \leq e.$$

Thus, $\mathrm{Q0}(\hat{e})$ holds after Step 5. If $\hat{e} > n + v$, then we claim that all the edges are now red by (4.35). Next, all edges are returned to black and the wrap-around stretches only edges (e,f) with

$$f < \hat{e} - n \leq e.$$

Since, by (4.35), there are no edges with tail vertices less than $\hat{e} - n < 0$, $\mathrm{Q0}(\hat{e} - n)$ holds after the wrap-around. So, if we define the ghost variable z to be $\hat{e}$ if no wrap-around is performed and $\hat{e} - n$ if a wrap-around is performed, then $\mathrm{Q0}(z)$ is a postcondition of Step 6. Finally, let (e', f') be the projection of $(ez[l], fz[l])$ selected in the i^{th} execution of Step 7. By the definition of f', it can be shown that $z < f'$ and that there does not exist a head vertex f such that $z \leq f < f'$. Thus, by the assignment of f' to $\hat{f}$ in Step 4, the claim is established.

Note that from the argument, we have shown that only black edges are stretched and red edges are reverted to black only when all edges are red. These observations are also valid for Procedure 3C. Furthermore, it should not be difficult to see that Procedure 3B increases the η-difference of a black edge by M_1 and that of a red edge by $M_1 + \delta$.

For Procedure 3C, we claim that $\mathrm{Q1}(\hat{e})$ is a precondition for Step 5 of that procedure where

$$\mathrm{Q1}(X) \equiv \text{"For all edges } (e,f) \text{ in G(B), } e < X \text{ implies } (e,f) \text{ is black and } e \geq X \text{ implies } (e,f) \text{ is red."}$$

The initial condition is established by recalling the definition of $(ex[0], fx[0])$; the rest of the proof is analogous to the previous case.

Next, note that except in Step 2 of Procedure 3C where edges not to the left of vertex w are colored red once without being stretched, in all other circumstances, an edge is colored red only after it has been stretched. Thus, Procedure 3C increases the η-difference of a red edge to the left of vertex w by $M_2 + \delta$ and the η-difference of any other edge by M_2.

Now, if we let $(\hat{e}, \hat{f})$ be the projection of ξ, then both $\mathrm{Q0}(\hat{f})$ and $\mathrm{Q1}(\hat{f})$ hold upon termination of Procedures 3A. Thus, all edges have been colored red the same number of times. In fact, recalling the program state when Procedure 3A is called,[10] it follows from

[10] Namely, an edge to the left of vertex w has an η-difference of M and one that is not has an η-difference of $M + \delta$.

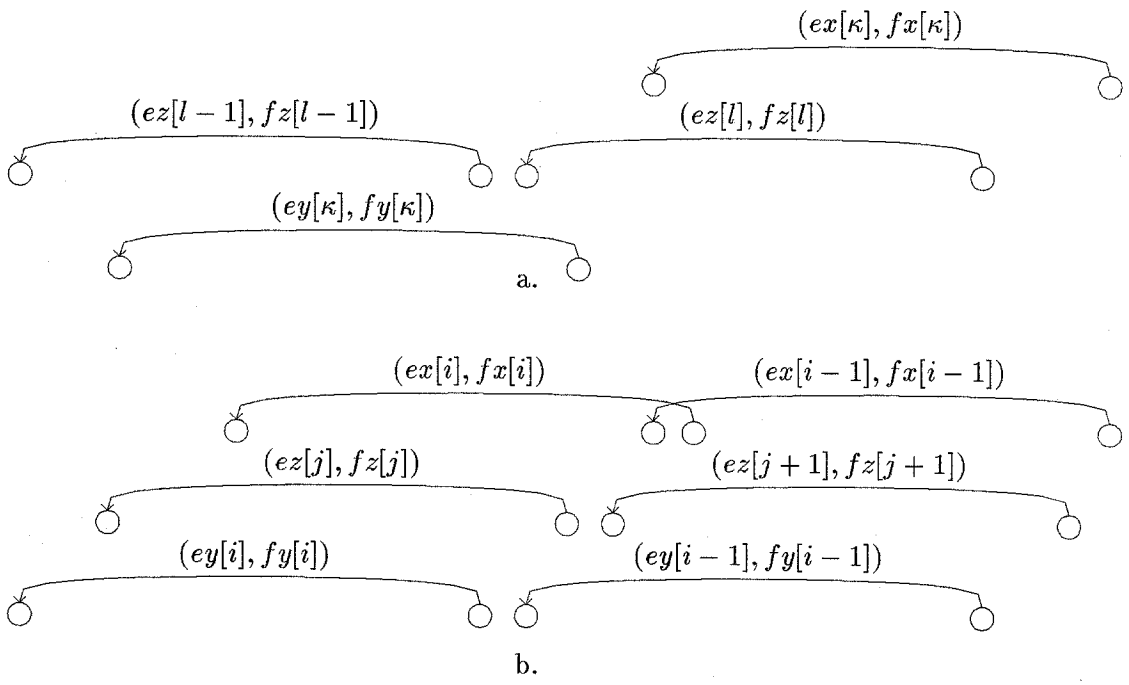


Figure 4.13: Termination of Procedure 3B

the above argument that after step 2 of the procedure, all edges have the same η-differences of $M + M_1 + M_2 + \delta$. Thus, invariant (4.33) is established by the assignment to M in Step 3 of Procedure 3A.

Furthermore, invariant (4.34) follows from the fact that T is updated for each δ placement and that a wrap-around is performed whenever a δ is placed where it upsets the constant phase length.

Now, it remains to show that the algorithm terminates; the crucial step is the termination of Procedure 3B. Let ρ_z be $\langle \ldots, (ez[l], fz[l]), \ldots, (ez[0], fz[0]) \rangle$ where $(ez[l], fz[l])$ is the leftmost edge of ρ_z that lies to the right of $(ey[\kappa], fy[\kappa])$. By the termination condition on Procedure 2A, $ey[\kappa] < fx[\kappa]$ as depicted in Figure 4.13a; in the figure, the three paths ρ_x, ρ_y and ρ_z are shown on separate rows so as to avoid confusion. Furthermore, $ez[l] \leq ex[\kappa]$ or else $(ez[l], fz[l])$ would not be the leftmost edge as specified by Step 6 of Procedure 3B. Now if $ez[l] = ex[\kappa]$, then Procedure 3B terminates. So, suppose $ey[\kappa] < ez[l] < ex[\kappa]$, or more generally, $ey[i] < ez[j] < ex[i]$. Then, as shown in Figure 4.13b, $ey[i-1]$ must be less than $ez[j+1]$ or else $(ez[j], fz[j])$ would replace $(ey[i], fy[i])$ in ρ_y. Also, $fx[i-1] > ez[j]$; if not, $(ez[j], fz[j])$ would replace $(ex[i], fx[i])$ in ρ_x. But, $fx[i-1] > ez[j]$ implies that $fz[j+1] \leq fx[i-1]$ by the definition of $fz[j+1]$. So, we have

$$ey[i] < ez[j] < ex[i] \Rightarrow ey[i-1] < ez[j+1] \leq ex[i-1].$$

Again, if $ez[j+1] = ex[i-1]$, then Procedure 3B terminates. But, if that's not the case, then, by induction, eventually there will exist $ez[\hat{\jmath}]$ such that

$$ey[0] < ez[\hat{\jmath}] < ex[0].$$

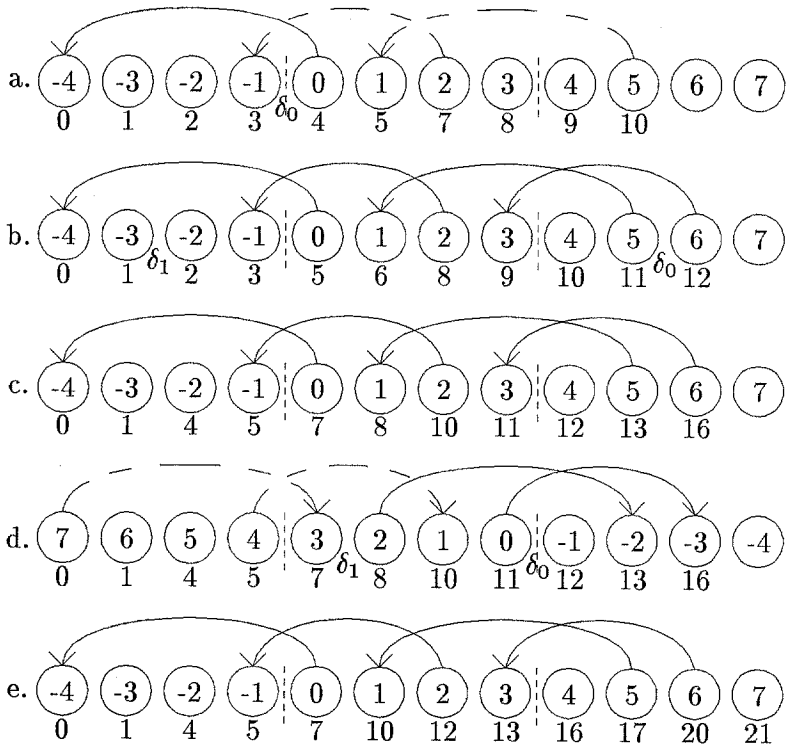


Figure 4.14: Application of Algorithm 3 for Example 4.14

But by the definitions of $ey[0]$ and $ex[0]$ in Procedure 2B, this is impossible due to the fact that $(ey[0], fy[0])$ is immediately to the left of $(ex[0], fx[0])$. This is the contradiction we seek to prove that Procedure 3B must terminate. By the exit condition on Procedure 3C, it also must terminate.

The correctness of Algorithm 3 in constructing $\{\eta(i) \mid h - n \leq i < n\}$, M, and T now follows directly. First, (4.30) is satisfied by the initial assignments of η-values and the fact that they are modified only by δ placements. Conditions (4.31) and (4.32) are consequences of the invariants (4.33) and (4.34).

The example below illustrates the operations that are performed by Algorithm 3.

Example 4.14: Let $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ with

$$\begin{aligned} \mathrm{A} &= \{\alpha_0, \alpha_1, \ldots, \alpha_7\}; \\ \Gamma &= \alpha_4\alpha_3\alpha_4\alpha_7\alpha_0\alpha_6(\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7)^\star; \\ \mathrm{C} &= \{(\alpha_0, \alpha_4), (\alpha_2, \alpha_7), (\alpha_5, \alpha_1), (\alpha_6, \alpha_3)\}. \end{aligned}$$

First, Algorithm 1 is applied to the leftmost 9 nodes yielding $M = 4$ and $T = 9$. For $\eta(5)$ we first assign it the value of $\eta(5 - n) + T = 10$. Now, since $\eta(5) - \eta(1) = 5 > M$, Procedure 3A is executed on B whose correspondence graph is shown in Figure 4.14a. Procedure 2A then generates ρ_x and ρ_y starting with nodes 5 and 2, respectively; the projections of the edges of ρ_x are shown as dashed arcs in Figure 4.14a. Next, Procedure 3B places δ_0 to the right of vertex -1. The next edge selected by the procedure is (5,1); since this is a projection of an edge in ρ_x, Procedure 3B sets ξ to (5,1) and stops. Procedure 3C does

not place any δ since the first edge it selects is ξ. The resultant configuration is shown in Figure 4.14b.

Next, for vertex 6, we first assign it a η-value of 12. However, the η-difference for the associated edge is $\eta(6) - \eta(3) = 3 < M = 5$. So, in this case, the single edge (6,3) should be stretched by $\delta = 2$. We do so immediately by placing a δ between vertices 5 and 6 and performing a wrap-around to maintain the constant phase length. See Figure 4.14c for the result; note that T has been incremented by δ due to these operations. Now, the edge (0,−4) has been stretched so all the edges to the *right* of it must have their η-differences incremented to compensate. The method for accomplishing this is to apply Procedures 3A but with the left and right senses interchanged as illustrated by Figure 4.14d. Next, Procedure 2A generates ρ_x and ρ_y starting with nodes 7 and 4, respectively. Procedure 3B then yields the placement of δ_0 shown in that figure. After δ_0, Procedure 3B selects the edge (4,1) which is a projection of an edge in ρ_x; thus, it sets ξ to (4,1) and terminates. Procedure 3C then places the final δ_1 and stops, since the next edge selected is (4,1). The final configuration is shown in Figure 4.14e. □

Now, all that remains to the proof of Theorem 4 is to generate a CSSF for $\mathcal{P}$, taking the occurrences in the head of Γ into account. Toward that end, we construct a consistent set of sequence steps for $\bar{\mathcal{P}} = \langle \bar{\mathrm{A}}, \bar{\Gamma}, \bar{\mathrm{C}} \rangle$, the two-phase expansion of $\mathcal{P}$, by modifying $\{\eta(i)\}$ to obtain a new set $\{\eta'(i) \mid 0 \leq i < n' + 2n\}$ which satisfies

$$\forall\, i : 0 \leq i < n' + 2n - 1 : \eta'(i) < \eta'(i+1), \tag{4.36}$$

$$\forall\, e, f : (\beta_e, \beta_f) \in \bar{\mathrm{C}} : \eta'(e) = \eta'(f) + M, \tag{4.37}$$

$$\forall\, i : n' \leq i < n' + n : \eta(i) = \eta(i+n) + T. \tag{4.38}$$

The construction is performed by Algorithm 4 which sets $\eta'(k)$ as k ranges from n' to 0 by maintaining the invariant

$$\forall\, e, f : (\beta_e, \beta_f) \in \bar{\mathrm{C}} \wedge k \leq f : \eta'(e) - \eta'(f) = M. \tag{4.39}$$

ALGORITHM 4:
Input: A pseudo-cycle canonical program $\mathcal{P}$ satisfying (4.18), (4.19), and (4.20).
Output: Integers T, M, and a set of integers $\{\eta'(i) \mid 0 \leq i < n' + 2n\}$ satisfying (4.36), (4.37), and (4.38).

1. Execute Algorithm 3 to obtain $\{\eta(i) \mid h - n \leq i < n\}$ and initial values of M and T. For i such that $0 \leq i < n$, set $\eta'(i + n')$ to $\eta(i)$ and $\eta'(i + n + n')$ to $\eta(i) + T$.

2. Initialize k to be n'.

3. If $k = 0$ then exit; else, decrement k and then set $\eta'(k)$ to $\eta'(k+1) - 1$.

4. If k is not a head vertex then go to step 3. Else, let k' be the tail vertex corresponding to k. If $\eta'(k') - \eta'(k) \leq M$, then decrease $\eta'(k)$, if necessary, to maintain (4.39) and go to step 3 afterward. If $\eta'(k') - \eta'(k) > M$, then execute the following steps.

5. Let $\hat{f}$ be the leftmost head vertex to the right of k. If no such vertex exists, let $\hat{f}$ be n'. Let m be the number of vertices between $\hat{f}$ and k.

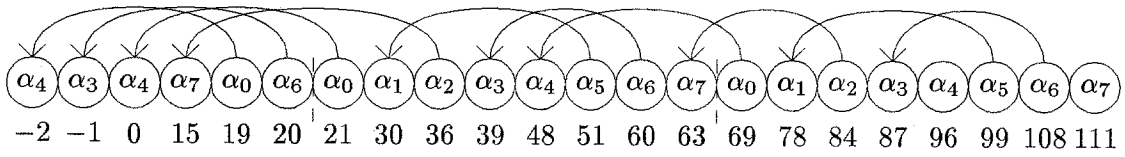


Figure 4.15: Application of Algorithm 4 for Example 4.15

6. Scale up the η'-values for all vertices not to the left of $\hat{f}$ by a factor of m. Also, assign to $\eta'(\hat{f}-1), \eta'(\hat{f}-2), \ldots, \eta'(k+1)$ values of $\eta'(\hat{f})-1, \eta'(\hat{f})-2, \ldots, \eta'(\hat{f})-\hat{f}+k+1$, respectively.

7. Due to (4.19), $k' > k$. So, set $\eta'(k)$ to $\eta'(k') - M \times m$. Also, update M to $M \times m$ and T to $T \times m$. Go back to Step 3.

By the correctness of Algorithm 3, (4.39) holds for the first execution of Step 3. It continues to hold for the next execution since, if k is a head vertex and $\eta'(k') - \eta'(k) > M$, then the η'-differences for all edges to the right of vertex k are scaled up by the same factor and $\eta'(k') - \eta'(k) = M$ after the assignments. Also, it should be clear that, for this case, the ordering is preserved among $\{\eta'(i) \mid k < i < n' + 2n\}$. Additionally, using the monotonicity condition on the head of Γ, it can be shown that $\eta'(k) < \eta'(k+1)$. Since $\eta'(i+n) - \eta'(i)$ is scaled up uniformly for all i such that $0 \leq i < n$, (4.38) is also satisfied. Thus, Algorithm 4 is correct.

With the construction of $\{\eta'(i) \mid 0 \leq i < n' + 2n\}$, $\sigma[l]$ is defined by

$$\sigma[l](\alpha_i, t) = \begin{cases} \eta'(\mathbf{ev}(\alpha_i, t)) + lM & \text{if } t < \mathbf{lag}(\alpha_i) \\ \eta'(n'+i) + lM + T(t - \mathbf{lag}(\alpha_i)) & \text{if } t \geq \mathbf{lag}(\alpha_i). \end{cases} \tag{4.40}$$

Then, for any N, the set $\{\sigma[l] \mid 0 \leq l < N\}$ is a CSSF for $\mathcal{P}$, and since $\sigma[l](\alpha_i, t+1) - \sigma[l](\alpha_i, t) = T$ holds for all but a finite number of (α_i, t), $\mathcal{P}$ has CRT.

Example 4.15: For the previous example, Algorithm 4 yields the assignments shown in Figure 4.15; the scaling up by $m = 3$ occurs when $k = 3$. Then, $\{\sigma[l] \mid 0 \leq l < N\}$, as defined below, can be verified to be a CSSF for $\mathcal{P}$:

$$\sigma[l](\alpha_0, t) = \begin{cases} 19 + 21l & \text{if t} = 0 \\ 21 + 21l + 48(t-1) & \text{if } t > 0, \end{cases} \quad \sigma[l](\alpha_1, t) = 30 + 21l + 48t,$$

$$\sigma[l](\alpha_3, t) = \begin{cases} -1 + 21l & \text{if t} = 0 \\ 39 + 21l + 48(t-1) & \text{if } t > 0, \end{cases} \quad \sigma[l](\alpha_2, t) = 36 + 21l + 48t,$$

$$\sigma[l](\alpha_4, t) = \begin{cases} -2 + 21l & \text{if t} = 0 \\ 9 + 21l + 48(t-1) & \text{if } t > 0, \end{cases} \quad \sigma[l](\alpha_5, t) = 51 + 21l + 48t,$$

$$\sigma[l](\alpha_6, t) = \begin{cases} 20 + 21l & \text{if } t = 0 \\ 60 + 21l + 48(t-1) & \text{if } t > 0, \end{cases} \quad \sigma[l](\alpha_7, t) = 15 + 21l + 48t.$$

Furthermore, since the difference in sequence steps between any two consecutive occurrences cannot be more than $T = 48$, $\mathcal{P}$ has CRT. □

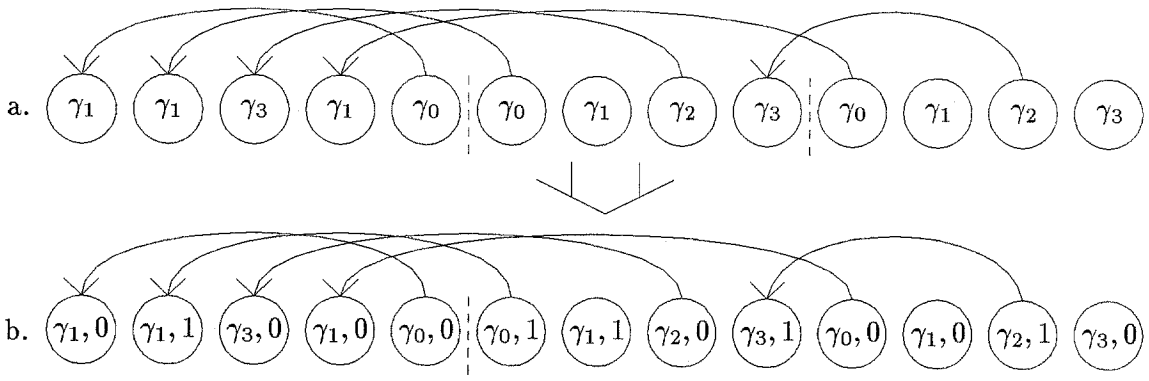


Figure 4.16: Program with Maximum Phase Difference of 2

4.7 Non-Canonical Programs

We'll now show how to transform a non-canonical program $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ that satisfies (4.9) into a canonical one. Without loss of generality, we can assume that $g = 0$, since we can always unroll the body of Γ if necessary. The steps to transform $\mathcal{P}$ into a canonical program are fairly simple and can be illustrated by the following examples.

Example 4.16: Let $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ with

$$\begin{aligned} \mathrm{A} &= \{\gamma_0, \gamma_1, \gamma_2, \gamma_3\}; \\ \Gamma &= \gamma_1\gamma_1\gamma_3\gamma_1\gamma_0(\gamma_0\gamma_1\gamma_2\gamma_3)^\star; \\ \mathrm{C} &= \{(\gamma_0, \gamma_1), (\gamma_2, \gamma_3)\}. \end{aligned}$$

The maximum phase difference of $\mathcal{P}$ is

$$\Delta(\gamma_0, \gamma_1) = \mathbf{lag}(\gamma_1) - \mathbf{lag}(\gamma_0) = 3 - 1 = 2$$

and so $\mathcal{P}$ is not canonical. Let $\Delta = 2$. We will now outline the general steps to transform a program with maximum phase difference of Δ into a canonical program $\mathcal{P}' = \langle \mathrm{A}', \Gamma', \mathrm{C}' \rangle$ which specifies the same communication behavior.

In the top part of Figure 4.16, the correspondence graph for the head and the first Δ phases of Γ is shown. Note that by (4.9), the correspondences for all the head vertices in the head of Γ are included since the maximum phase difference is Δ. Initially, the vertices are labeled with the corresponding actions. Our approach is to replace the these labels by new symbols from the set $\mathrm{A}' = \mathrm{A} \times \{0, 1, \ldots, \Delta - 1\}$ using the following transformation:

> For every vertex v with label γ_i, replace the label with $(\gamma_i, j \bmod \Delta)$, where j is the number of vertices to the left of vertex v with old labels γ_i.

The result for this example is shown in the bottom of Figure 4.16.

The head of Γ' will be the same as the head of Γ (after the relabeling), and the first phase in the body of Γ' will be the first Δ phases in the body of Γ. Thus,

$$\Gamma' = (\gamma_1, 0)(\gamma_1, 1)(\gamma_3, 0)(\gamma_1, 0)(\gamma_0, 0)(\mathbf{c})^\star,$$

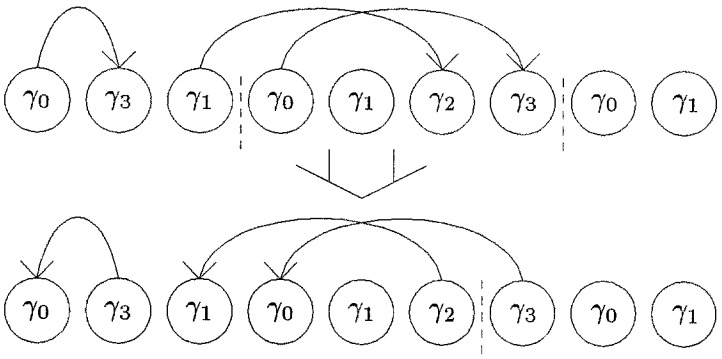


Figure 4.17: Program with Maximum Phase Difference of 0

where

$$\mathbf{c} = (\gamma_0, 1)(\gamma_1, 1)(\gamma_2, 0)(\gamma_3, 1)(\gamma_0, 0)(\gamma_1, 0)(\gamma_2, 1)(\gamma_3, 0).$$

Next, we define a new set of correspondences as

$$\mathrm{C}' = \{((\gamma_e, i), (\gamma_f, i)) \mid (\gamma_e, \gamma_f) \in \mathrm{C} \wedge 0 \leq i < \Delta\}.$$

Then, $\mathcal{P}'$ is canonical because Δ phases in Γ is represented as a single phase is Γ'. Furthermore, it can be shown that the communication behaviors of $\mathcal{P}$ and $\mathcal{P}'$ are the same since they are just different ways of representing the same program. □

It now remains to show that any program with non-positive maximum phase difference can be transformed into one with a positive maximum phase difference. To accomplish this, we make use of the concept of duals introduced at the end of Chapter 2.

Example 4.17: Let $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ with

$$\begin{aligned} \mathrm{A} &= \{\gamma_0, \gamma_1, \gamma_2, \gamma_3\}; \\ \Gamma &= \gamma_0\gamma_3\gamma_1(\gamma_0\gamma_1\gamma_2\gamma_3)^*; \\ \mathrm{C} &= \{(\gamma_0, \gamma_3), (\gamma_1, \gamma_2)\}. \end{aligned}$$

The maximum phase difference of $\mathcal{P}$ is

$$\Delta(\gamma_0, \gamma_3) = \mathbf{lag}(\gamma_3) - \mathbf{lag}(\gamma_0) = 1 - 1 = 0$$

and so it is not canonical. Consider the dual of this program, namely, $\mathcal{P}^\perp = \langle \mathrm{A}, \Gamma^\perp, \mathrm{C}^\perp \rangle$ where

$$\mathrm{C}^\perp = \{(\gamma_3, \gamma_0), (\gamma_2, \gamma_1)\}.$$

Furthermore, we unroll Γ so that γ_3 becomes the first symbol in the body of $\Gamma^\perp$:

$$\Gamma^\perp = \gamma_0\gamma_3\gamma_1\gamma_0\gamma_1\gamma_2(\gamma_3\gamma_0\gamma_1\gamma_2)^*.$$

See Figure 4.17. Note that the maximum phase difference of $\mathcal{P}^\perp$ is 1 and it is therefore canonical. □

In general, we unroll Γ so that γ_h in $\mathcal{P}$ becomes the first symbol in the body of $\Gamma^\perp$. Let $\Delta(\gamma_g, \gamma_h) = \Delta \leq 0$ in $\mathcal{P}$, then $\Delta(\gamma_h, \gamma_g) = 1 - \Delta > 0$ in $\mathcal{P}$ due to the definition of $\mathrm{C}^\perp$ and the fact that the extension of the head of $\Gamma^\perp$ does not get past γ_h as illustrated by the example above. Thus, the maximum phase difference of $\mathcal{P}^\perp$ is positive and we can conclude that any pseudo-cyclic program satisfying (4.9) can be transformed into a canonical one.

4.8 Concluding Remarks

Since any deadlock-free program can be transformed into a canonical one, by Theorem 3 and Theorem 4, we have established criteria for checking the communication behavior of a general class of programs. Currently, checking for CRT involves the application of an algorithm whose worst-case time complexity is of the order $|C|^3$, though, for the examples we have dealt with, the actual performance is much better. We believe that it may be possible to come up with a faster method. Furthermore, the other necessary conditions for CRT — (4.18), (4.19), (4.21), (4.23), and (4.24) — are easy to check and can be used to rule out most of the candidate programs that do not have CRT.

Chapter 5

Infinite-Slack Programs

5.1 Definitions

So far in this paper, we have only restricted our analyses to zero-slack communications. We can generalize some of the results to communications with one-sided infinite slacks, *i.e.*, for a given channel, the number of communications completed on one of its ports (called the **input** port) is no greater than (but not necessarily identical to) the number of communications completed on the other port (call the **output** port). Thus, unlike zero-slack communications, a process cannot be suspended while performing a communication action on an output port. Since there are now two possible types of correspondences between processes $p[l]$ and $p[l+1]$, depending on whether $p[l]$ contains the input or output port, two sets of correspondences are included in the formalization of an infinite-slack program.

Definition 5.1 *An infinite-slack* **program** $\mathcal{P}$ *is a quadraple* $\langle \mathrm{A}, \Gamma, \mathrm{C}_{oi}, \mathrm{C}_{io} \rangle$ *where*

- $\mathrm{A} = \{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$ *is called the* **alphabet** *of* $\mathcal{P}$*, and each* α_i *is a* **symbol** *in the alphabet;*
- Γ *is a (possibly non-terminating) sequence of symbols of* A. Γ *is called the* **communication pattern** *of* $\mathcal{P}$*;*
- $\mathrm{C}_{oi} \subseteq \mathrm{A} \times \mathrm{A}$ *is the* **set of output-input correspondences** *for* $\mathcal{P}$*;*
- $\mathrm{C}_{io} \subseteq \mathrm{A} \times \mathrm{A}$ *is the* **set of input-output correspondences** *for* $\mathcal{P}$.

The definition of CSSF is then modified to the one below to reflect that an input action cannot precede the matching output action.

Definition 5.2 *For a given program* $\mathcal{P}$ *and a positive integer* N*, the set of functions* $\{\sigma[l] \mid 0 \leq l < N\}$ *is a CSSF if*

$$\forall\, l : 0 \leq l < N : \sigma[l] \text{ is a sequence function for } \mathcal{P}, \tag{5.1}$$

$$\forall\, e, f, l, t : (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} \wedge 0 < l < N \wedge t \geq 0 : \sigma[l-1](\alpha_e, t) \leq \sigma[l](\alpha_f, t), \tag{5.2}$$

$$\forall\, e, f, l, t : (\alpha_e, \alpha_f) \in \mathrm{C}_{io} \wedge 0 < l < N \wedge t \geq 0 : \sigma[l-1](\alpha_e, t) \geq \sigma[l](\alpha_f, t). \tag{5.3}$$

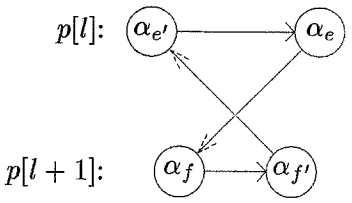


Figure 5.1: Violation of Eqn. (5.6)

Notice that a zero-slack correspondence (α_e, α_f) can be specified under this model by specifying that (α_e, α_f) belongs to both C_{oi} and C_{io}. Finally, the concepts of absense of deadlocks, constant response time, dual programs[1], cyclic programs, and pseudo-cyclic programs, are extended in a natural manner to infinite-slack programs. Also, the restrictions on correspondences mentioned in Section 2.5 become

$$\forall\, e, f, e', f' : (\alpha_e, \alpha_f), (\alpha_{e'}, \alpha_{f'}) \in C_{oi} \cup C_{io} : e = e' \Leftrightarrow f = f', \tag{5.4}$$

$$\forall\, e, f, e', f' : (\alpha_e, \alpha_f), (\alpha_{e'}, \alpha_{f'}) \in C_{oi} \cup C_{io} : f \neq e'. \tag{5.5}$$

5.2 Cyclic Programs

Theorem 5 *A cyclic infinite-slack program $\mathcal{P}$ is deadlock-free, if and only if it satisfies*

$$(\nexists\, e, f, e', f' : (\alpha_e, \alpha_f) \in C_{oi} \wedge (\alpha_{e'}, \alpha_{f'}) \in C_{io} : e' < e \wedge f < f'. \tag{5.6}$$

To see that this requirement is necessary, consider its violation as depicted in Figure 5.1. Since $(\alpha_e, \alpha_f) \in C_{oi}$, the occurrence of α_e in $p[l]$ must coincide or precede α_f in $p[l+1]$. Thus, an arrow with a broken head is used to indicate this precedence relationship. Similarly, $\alpha_{e'}$ in $p[l]$ must coincide or follow $\alpha_{f'}$ in $p[l+1]$. Thus, a cycle of dependencies forms if (5.6) is violated and $\mathcal{P}$ has a deadlock.

We will next show that (5.6) is also sufficient for a cyclic program to be deadlock-free. The approach is analoguous to the one taken for zero-slack programs. For any N, we assign to each symbol α_i in process $p[l]$ an integer $\tau[l](i)$ satisfying:

$$\forall\, l, i : 0 \leq l < N \wedge 0 \leq i < n-1 : \tau[l](i) < \tau[l](i+1), \tag{5.7}$$

$$\forall\, l, e, f : 0 < l < N \wedge (\alpha_e, \alpha_f) \in C_{oi} : \tau[l-1](e) \leq \tau[l](f), \tag{5.8}$$

$$\forall\, l, e, f : 0 < l < N \wedge (\alpha_e, \alpha_f) \in C_{io} : \tau[l-1](e) \geq \tau[l](f). \tag{5.9}$$

Then, define T_N and $\{\sigma[l] \mid 0 \leq l < N\}$ as per (3.6) and (3.7) to obtain a CSSF for $\mathcal{P}$. The algorithm to construct $\{\tau[l](i)\}$ is given below:

ALGORITHM 5:
Input: Cyclic infinite-slack program $\mathcal{P}$ satisfying (5.6) and integer N.
Output: Set of integers $\{\tau[l](i)\}$ satisfying (5.7), (5.8), and (5.9).

1. Set $N' = 1$. Set $\tau[0](i)$ to i.

2. If $N' = N$, exits; else, continue.

[1] $C_{io}^{\perp} = \{(\alpha_e, \alpha_f) \mid (\alpha_f, \alpha_e) \in C_{oi}\}$ and $C_{oi}^{\perp} = \{(\alpha_e, \alpha_f) \mid (\alpha_f, \alpha_e) \in C_{io}\}$.

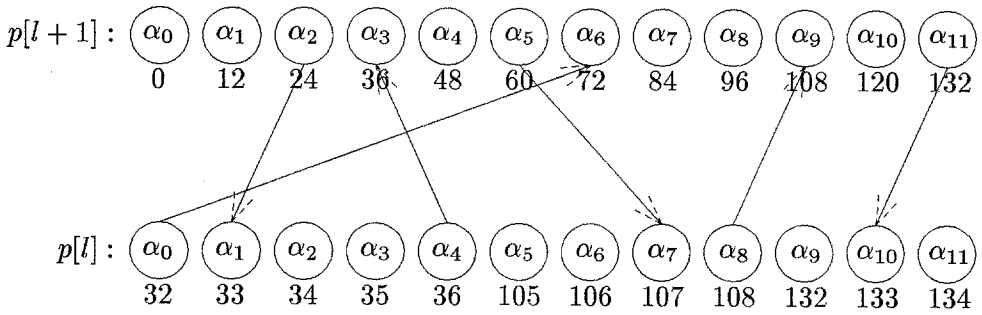


Figure 5.2: Construction of $\{\tau[l](i)\}$ for Example 5.1

3. Increment N' and apply Procedure 5A.

4. Go back to Step 2.

PROCEDURE 5A:

1. Multiply each element of the set $\{\tau[l](i) \mid 0 \leq l < N' - 1 \wedge\ 0 \leq i < n\}$ by n.

2. Set k to -1.

3. If the set $S = \{e \mid \exists f :: (\alpha_e, \alpha_f) \in \mathrm{C}_{io} \ \wedge\ f > k\}$ is empty then go to Step 7. Else, let $\hat{e}$ be the smallest element in S and $\hat{f}$ be its corresponding action, i.e., $(\alpha_{\hat{e}}, \alpha_{\hat{f}}) \in \mathrm{C}_{io}$.

4. Assign $\tau[N' - 2](\hat{e})$ to $\tau[N' - 1](\hat{f})$.

5. For i such that $k < i < \hat{f}$, assign to $\tau[N' - 1](i)$ the maximum value that still maintains the order $\tau[N' - 1](i) < \tau[N' - 1](i + 1)$.

6. Let k be $\hat{f}$ and go back to Step 3.

7. For i such that $k < i < n$, assign the minimum value to $\tau[N' - 1](i)$ so that $\tau[N' - 1](i) < \tau[N' - 1](i + 1)$ is maintained and $\tau[N' - 1](i) \geq \tau[N' - 2](n - 1)$.

Example 5.1: Let $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C}_{oi}, \mathrm{C}_{io} \rangle$ where

$$\begin{aligned} \mathrm{A} &= \{\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{11}\}; \\ \Gamma &= (\alpha_0\alpha_1\alpha_2 \ldots \alpha_{11})^\star; \\ \mathrm{C}_{oi} &= \{(\alpha_2, \alpha_1), (\alpha_5, \alpha_7), (\alpha_{11}, \alpha_{10})\}; \\ \mathrm{C}_{io} &= \{(\alpha_6, \alpha_0), (\alpha_3, \alpha_4), (\alpha_9, \alpha_8)\}. \end{aligned}$$

Figure 5.2 graphically depicts the correspondences.

For $N = 1$, we get $\tau[0](i) = i$. Next, we will trace through the assignments performed in Procedure 5A for $N = 2$. The first $(\alpha_{\hat{e}}, \alpha_{\hat{f}})$ is (α_3, α_4) and we assign values to $\tau[1](0)$ to $\tau[1](4)$ as shown. The next time through the loop yields $(\alpha_{\hat{e}}, \alpha_{\hat{f}}) = (\alpha_9, \alpha_8)$ which leads to the assignments for $\tau[1](5)$ to $\tau[1](8)$. Afterward, S in Step 3 becomes empty and the rest of the $\tau[1](i)$ are assigned values as prescribed in Step 7. □

The correctness of the algorithm hinges on establishing the following loop invariants for Procedure 5A:

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{io} \ \wedge f \leq k : \tau[N'-2](e) \geq \tau[N'-1](f), \tag{5.10}$$

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} \ \wedge f \leq k : \tau[N'-2](e) \leq \tau[N'-1](f). \tag{5.11}$$

These hold initially since $k = (-1)$. To see that (5.10) is a loop invariant, recall the definition of $\hat{e}$ as the minimum member of the set S. For (5.11), note that after Step 3, if $(\alpha_e, \alpha_f) \in \mathrm{C}_{oi}$ and $k < f < \hat{f}$, then, by (5.6) and (5.4), $e < \hat{e}$. Then, by the assignments in Step 5 and the fact that $\tau[N'-2](\hat{e}-1) - \tau[N'-2](\hat{e})$ must be at least n, we have $\tau[N'-2](e) \leq \tau[N'-2](\hat{e}-1) < \tau[N'-1](f)$. Thus, both (5.10) and (5.11) hold when the loop is exited. Finally, the assignments in Step 7 ensures that for any $(\alpha_e, \alpha_f) \in \mathrm{C}_{oi}$ with $f > k$, $\tau[N'-2](e) \leq \tau[N'-2](n-1) \leq \tau[N'-1](f)$. The rest of the argument for showing the correctness of Algorithm 5 is straightforward and will be omitted.

For CRT, we have the following result:

Theorem 6 *A cyclic infinite-slack program $\mathcal{P}$ has CRT if and only if it satisfies (5.6) and*

$$(\nexists\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} : f < e) \vee (\nexists\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{io} : e < f). \tag{5.12}$$

The necessity of (5.6) is obvious. Also, it is easy to see that if (5.12) is violated, a situation like the one depicted on Figure 3.3c occurs and no CRT is possible. To establish the sufficiency of (5.6) and (5.12), we attempt to produce a constant M and a set of integers $\{\pi(i) \mid 0 \leq i < n\}$ such that

$$\forall\, i : 0 \leq i < n-1 : \pi(i) < \pi(i+1), \tag{5.13}$$

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} : \pi(e) \leq \pi(f) + M, \tag{5.14}$$

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{io} : \pi(e) \geq \pi(f) + M. \tag{5.15}$$

Once these values are obtained, we can define T and $\{\sigma[l] \mid 0 \leq l < N\}$ as per (3.11) and (3.12) to obtain a CSSF for $\mathcal{P}$ that induces CRT.

The actual construction of M and $\{\pi(i) \mid 0 \leq i < n\}$ when $\mathcal{P}$ satisfies

$$\nexists\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{io} : e < f \tag{5.16}$$

will be described by Algorithm 6 below. If, instead, $\mathcal{P}$ satisfies

$$\nexists\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} : f < e \tag{5.17}$$

then we can replace $\mathcal{P}$ with its dual which satsifes (5.16).

Observe that if $(\alpha_e, \alpha_f) \in \mathrm{C}_{oi} \ \wedge \ e < f$, then, if (5.13) holds, $\pi(e) < \pi(f) + M$ automatically. Thus, the first step of Algorithm 6 removes all such correspondences from consideration.

As in Algorithm 1, we make use of a correspondence graph to facilitate the discussion. There are now two types of arcs for the graph, representing the two types of correspondences. By convention, arcs due to input-output correspondences are placed on top of the nodes while those due to output-input correspondences are placed below. See next example for an illustration.

Algorithm 6 systematically defines $\pi(k)$ as k ranges from 0 to $n-1$. It attempts to construct $\{\pi(i) \mid 0 \leq i < n\}$ by enforcing the proper ordering within $\{\pi(i) \mid 0 \leq i \leq k\}$ and maintaining the following loop invariants:

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{B}_{oi} \wedge e \leq k : \pi(e) - \pi(f) \leq M, \tag{5.18}$$

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{io} \wedge e \leq k : \pi(e) - \pi(f) \geq M. \tag{5.19}$$

Graphically, this invariant translates into requiring that all the bottom (top) arcs not to the right vertex k have π-difference at most (at least) M.

ALGORITHM 6:
Input: Cyclic infinite-slack program $\mathcal{P}$ satisfying (5.6), (5.12), and (5.16).
Output: Integer M and set of integers $\{\pi(i) \mid 0 \leq i < n\}$ satisfying (5.13), (5.14), and (5.15).

1. Let $\mathrm{B}_{oi} = \{(\alpha_e, \alpha_f) \mid (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} \ \wedge \ f < e\}$.
2. Initialize M, $\pi(0)$, and k to 0.
3. If $k = n-1$, then exit algorithm.
4. Increment k and set $\pi(k) = \pi(k-1) + 1$.
5. If k is not a tail vertex then go to Step 3. Else, let k' be the corresponding head vertex.
6. If k is a tail vertex of a top arc and the π-difference is less than M, then maintain (5.19) by increasing $\pi(k)$ to $\pi(k') + M$.
7. If k is a tail vertex of a bottom arc and the π-difference is greater than M, then let $\delta = \pi(k) - \pi(k') - M$. Next, execute Procedure 6A so as to increase the π-differences for all top arcs to the left of vertex k by δ in order to maintain (5.19). During the application of the procedure, all arcs to the right of vertex k are ignored.
8. Go to Step 3.

PROCEDURE 6A:

1. Initialize x to $k-1$.
2. Find $\hat{f}$, the rightmost head vertex of a top arc such that its corresponding tail vertex, $\hat{e}$, is not to the right of vertex x. If no such vertex exists, then set M to $M + \delta$ and exit procedure.
3. Increase the π-values for all vertices to the right of $\hat{f}$ by δ.
4. Set x to $\hat{f}$ and go to Step 2.

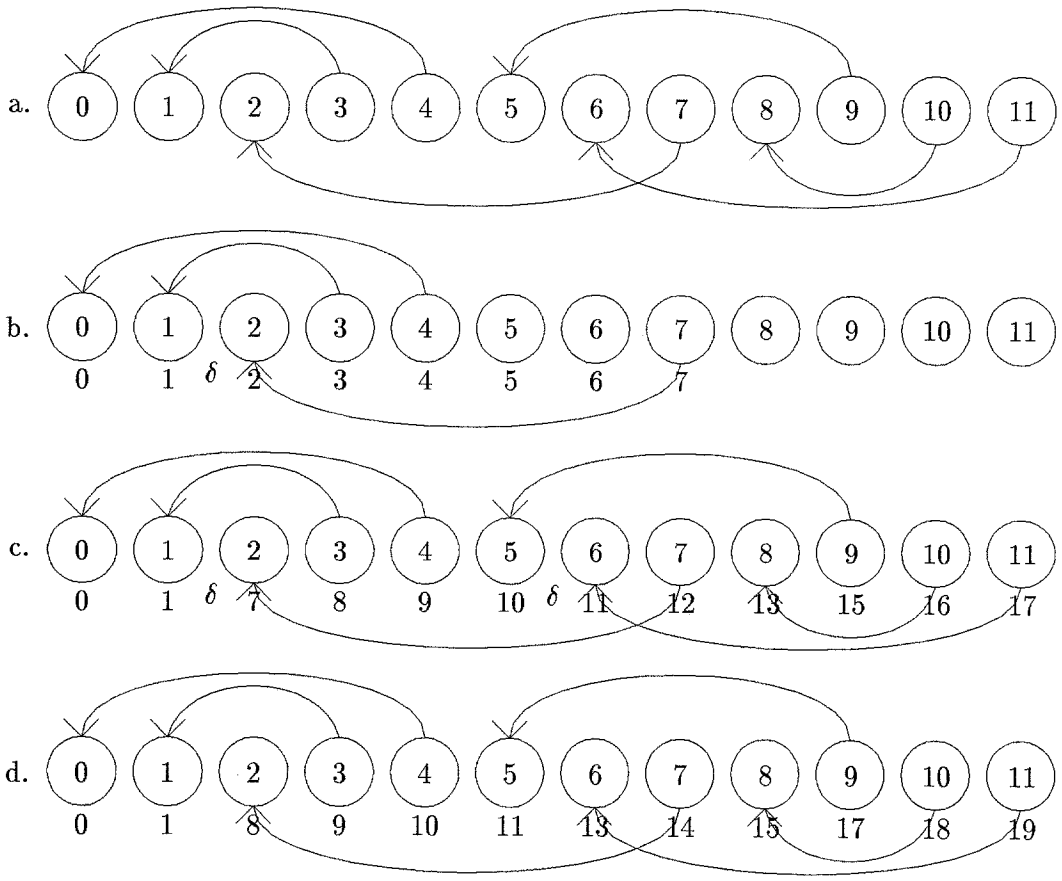


Figure 5.3: Application of Algorithm 6 for Example 5.2

Example 5.2: Let $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C} \rangle$ with

$$\begin{aligned} \mathrm{A} &= \{\alpha_0, \alpha_1, \ldots, \alpha_{11}\}, \\ \Gamma &= (\alpha_0\alpha_1 \ldots \alpha_{11})^\star, \\ \mathrm{C}_{oi} &= \{(\alpha_7, \alpha_2), (\alpha_{10}, \alpha_8), (\alpha_{11}, \alpha_6)\}, \\ \mathrm{C}_{io} &= \{(\alpha_4, \alpha_0), (\alpha_3, \alpha_1), (\alpha_9, \alpha_5)\}. \end{aligned}$$

The correspondence graph of $\mathcal{P}$ is shown in Figure 5.3a. The operations performed by Algorithm 6 are shown below and illustrated in the rest of Figure 5.3.

1. Initially, $\mathrm{B}_{oi} = \mathrm{C}_{oi}$, $M = 0$, $\pi(0) = 0$, and $k = 0$.

2. $\pi(i)$ becomes i for $1 \leq i < 6$.

3. $\pi(7)$ becomes 7. See Figure 5.3b. Since vertex 7 is a tail vertex of a bottom arc and $\pi(7) - \pi(2) = 5 > M = 0$, we set δ to 5. Procedure 6A is to be applied resulting in the placement of δ between vertices 1 and 2. The π-values after the update are shown in Figure 5.3c. M is now 5.

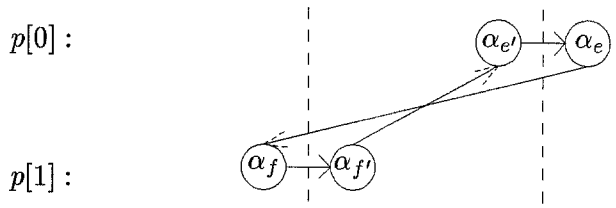


Figure 5.4: Violation of Eqn. (5.20)

4. $\pi(8)$ becomes 13.

5. $\pi(9)$ becomes 14. But since vertex 9 is the tail of a top arc whose π-difference is less than M, $\pi(9)$ is increased to 15 to maintain (5.19).

6. $\pi(10)$ becomes 16. No modification is necessary since the pi-difference of the bottom arc with tail vertex 16 is already less than M.

7. $\pi(11)$ becomes 17. The π-difference of the bottom arc with tail vertex 17 is 6 which is greater than M. So, δ is set to 1 and Procedure 6A places the δ's as shown in Figure 5.3c and assigns 5 to M. The final $\{\pi(i) \mid 0 \leq i < n\}$ is shown in Figure 5.3d.

□

The purpose Procedure 6A is to place δ's so that all top arcs not to the right of vertex k are stretched at least once. Note that this is sufficient to maintain the loop invariants (5.18) and (5.19) for Algorithm 6. The rest of the proof for the correctness of the algorithm is straightforward and is omitted.

Before ending this section, we like to remark that Theorems 5 and 6 completely characterize cyclic infinite-slack programs. Furthermore, Theorems 1 and 2 are direct corollaries of these two results.

5.3 Pseudo-Cyclic Programs

For pseudo-cyclic infinite-slack programs, we maintain the definitions of "lags", "event numbers" and "phase difference." As in the zero-slack case, there is a restriction on how much the phase differences of the correspondences for a given program can vary:

Lemma 5.1 *If a program $\mathcal{P}$ is deadlock-free, then*

$$\nexists\, e, f, e', f' : (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} \wedge (\alpha_{e'}, \alpha_{f'}) \in \mathrm{C}_{io} : \Delta(\alpha_e, \alpha_f) - \Delta(\alpha_{e'}, \alpha_{f'}) > 1. \quad (5.20)$$

The deadlock that results from the violation of (5.20) is shown in Figure 5.4. We can show that any program satisfying (5.20) can be transformed into a canonical one which is defined by the following:

Definition 5.3 *A program $\mathcal{P}$ is called* **canonical** *if its set of correspondence satisfy*

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{io} : \Delta(\alpha_e, \alpha_f) \geq 0; \quad (5.21)$$

$$\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} : \Delta(\alpha_e, \alpha_f) \leq 1. \quad (5.22)$$

If a program $\mathcal{P}$ does not satisfy the first condition, then, by (5.20), $\forall\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} : \Delta(\alpha_e, \alpha_f) \leq 0$. So, replace $\mathcal{P}$ by $\mathcal{P}^{\perp}$ which will then satisfies (5.21). Furthermore, if a deadlock-free program satisfies (5.21), then, it can be transformed into one that satisfies (5.22) as well by, if necessary, increasing the number of symbols per phase as done in the first example of Section 4.7.

Next, we extend Def. 4.7 in an obvious manner to infinite-slack programs: *i.e.*, the two-phase expansion of $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C}_{oi}, \mathrm{C}_{io} \rangle$ is another program $\bar{\mathcal{P}} = \langle \bar{\mathrm{A}}, \bar{\Gamma}, \bar{\mathrm{C}_{oi}}, \bar{\mathrm{C}_{io}} \rangle$ which contains all occurrences and correspondences within the head and the first two phases of Γ. Then, we have the following result:

Theorem 7 *Let $\mathcal{P}$ be a canonical infinite-slack program and $\bar{\mathcal{P}} = \langle \bar{\mathrm{A}}, \bar{\Gamma}, \bar{\mathrm{C}_{oi}}, \bar{\mathrm{C}_{io}} \rangle$ be its two-phase expansion. Then, $\mathcal{P}$ is deadlock-free if and only if $\bar{\mathcal{P}}$ satisfies*

$$\nexists\, e, f, e', f' : (\beta_e, \beta_f) \in \bar{\mathrm{C}_{oi}} \wedge (\beta_{e'}, \beta_{f'}) \in \bar{\mathrm{C}_{io}} : e' < e \wedge f < f'. \tag{5.23}$$

The necessity of this condition should be obvious since otherwise a cycle of dependencies like the one in Figure 5.1 would develop. Its sufficiency can be established by constructing integer T_N and a set of integers $\{\tau[l](i) \mid 0 \leq l < N \wedge 0 \leq i < n' + 2n\}$ such that

$$\forall\, l, i : 0 \leq l < N \wedge 0 \leq i < n' + 2n - 1 : \tau[l](i) < \tau[l](i+1), \tag{5.24}$$

$$\forall\, l, e, f : 0 < l < N \wedge (\beta_e, \beta_f) \in \bar{\mathrm{C}_{oi}} : \tau[l-1](e) \leq \tau[l](f), \tag{5.25}$$

$$\forall\, l, e, f : 0 < l < N \wedge (\beta_e, \beta_f) \in \bar{\mathrm{C}_{io}} : \tau[l-1](e) \geq \tau[l](f), \tag{5.26}$$

$$\forall\, l, i : 0 \leq l < N \wedge n' \leq i < n' + n : \tau[l](i+n) = \tau[l](i) + T_N, \tag{5.27}$$

$$\forall\, l, i : 0 \leq l < N : \tau[l-1](n' + n - 1) \leq \tau[l](n') + 2T_N, \tag{5.28}$$

$$\forall\, l, i : 0 \leq l < N : \tau[l](n' + n - 1) \leq \tau[l-1](n') + 2T_N, \tag{5.29}$$

and then defining a CSSF by (4.14). Note that though the steady-state behavior of a correspondence $(\alpha_e, \alpha_f) \in \mathrm{C}_{oi}$ with $\Delta(\alpha_e, \alpha_f) \leq -2$ is not included in the two-phase expansion, (5.3) is nevertheless satisfied for that correspondence provided (5.28) is valid. Similarly, (5.29) ensures that (5.3) is satisfied by $(\alpha_e, \alpha_f) \in \mathrm{C}_{io}$ with $\Delta(\alpha_e, \alpha_f) \geq 2$.

The actual construction is performed by Algorithm 7 in the Appendix. Its correctness can be proven by verifying that the above six conditions (with N replaced by N') are maintained by each execution of Procedure 7A. The most difficult part of the proof involves establishing (5.25) and (5.26). Toward that end, consider the predicate

$$\begin{aligned} &((\beta_e, \beta_f) \in \bar{\mathrm{C}_{io}} \Rightarrow \tau[N'-2](e) \geq \tau[N'-1](f)) \\ \wedge\ &((\beta_e, \beta_f) \in \bar{\mathrm{C}_{oi}} \Rightarrow \tau[N'-2](e) \leq \tau[N'-1](f)). \end{aligned}$$

This predicate holds for all (β_e, β_f) with $k < f < n' + n \wedge e < n' + n$ in the loop of Procedure 7A. After the assignment to $\tau[N'-1](i)$ for $k < i < n' + n$, it holds for all (β_e, β_f) with $f < n' + n$. Finally, upon exit from the procedure, it holds for all (β_e, β_f) due to the fact that if $f \geq n' + n$, then either $(\beta_{e-n}, \beta_{f-n})$ is also a correspondence or $(\alpha_{e'}, \alpha_{f'}) \in \mathrm{C}_{oi} \wedge \Delta(\alpha_{e'}, \alpha_{f'}) \leq -1$ where $\alpha_{f'} = \alpha_{f-n'-n}$ is the symbol in $\mathcal{A}$ represented by β_f and $\alpha_{e'}$ is its corresponding symbol. From this observation, (5.25) and (5.26) follow. The other four conditions — (5.24), (5.27), (5.28), and (5.29) — can also be readily established.

We now turn our attention to the determination of which pseudo-cyclic infinite-slack programs have CRT. Like the zero-slack case, we need the concept of cycles as defined below:

Definition 5.4 *For a canonical program $\mathcal{P}$, the* **sets of steady-state output-input correspondences** *is*

$$\mathrm{C}^{\circ}_{oi} = \{(e, f - n\Delta(\alpha_e, \alpha_f)) \mid (\alpha_e, \alpha_f) \in \mathrm{C}_{oi}\}$$

and the **extended set of steady-state output-input correspondences** *is*

$$\mathrm{C}^{\star}_{oi} = \{(e, f) \mid \exists\, j, \hat{e}, \hat{f} : j \in \mathbf{Z} \wedge (\hat{e}, \hat{f}) \in \mathrm{C}^{\circ}_{oi} : e = \hat{e} + nj \wedge f = \hat{f} + nj\}.$$

The definitions of C°_{io} and $\mathrm{C}^{\star}_{io}$ are analogous.

Definition 5.5 *For a program $\mathcal{P}$ with extended sets of steady-state correspondences $\mathrm{C}^{\star}_{oi}$ and $\mathrm{C}^{\star}_{io}$, a* **path** *ρ is a list*

$$\langle (e_0, f_0), (e_1, f_1), \ldots, (e_L, f_L) \rangle$$

such that $L > 0$ and $\forall\, i : 0 \le i \le L : (e_i, f_i) \in \mathrm{C}^{\star}_{oi} \cup \mathrm{C}^{\star}_{io}$. L is called the **length** *of the path.*

If $((\forall\, i : 0 \le i \le L : (e_i, f_i) \in \mathrm{C}^{\star}_{io}) \wedge (\forall\, i : 0 \le i < L : e_{i+1} < f_i))$, then the path is called a **negative path**. *If $((\forall\, i : 0 \le i \le L : (e_i, f_i) \in \mathrm{C}^{\star}_{oi}) \wedge (\forall\, i : 0 \le i < L : e_{i+1} > f_i))$, then the path is called a* **positive path**. *If $e_0 = e_L + nD$, then the path is called a* **cycle** *with* **period** *D.*

Next, we have the following theorem describing the *necessary* conditions for a program to have CRT.

Theorem 8 *If a canonical pseudo-cyclic infinite-slack program $\mathcal{P} = \langle \mathrm{A}, \Gamma, \mathrm{C}_{oi}, \mathrm{C}_{io} \rangle$ with two-phase expansion $\bar{\mathcal{P}} = \langle \bar{\mathrm{A}}, \bar{\Gamma}, \bar{\mathrm{C}}_{oi}, \bar{\mathrm{C}}_{io} \rangle$ has CRT, then it satisfies the followings:*

$$\mathcal{P} \text{ is deadlock-free,} \tag{5.30}$$

$$\bar{\mathcal{P}} \text{ satisfies } (\nexists\, e, f : (\beta_e, \beta_f) \in \bar{\mathrm{C}}_{oi} : f < e) \vee (\nexists\, e, f : (\beta_e, \beta_f) \in \bar{\mathrm{C}}_{io} : e < f), \tag{5.31}$$

$$\text{There does not exist a positive cycle and a negative cycle in } \mathcal{P} \text{ with the same length and period.} \tag{5.32}$$

The necessity of the first condition is obvious. If the second condition is violated, then a situation like the one depicted in Figure 3.3c exists in the two-phase expansion of $\mathcal{P}$ and therefore it does not have CRT. Finally, it is not difficult to see that if $\mathcal{P}$ has a negative cycle of length L and period D, then there exists an action α_j such that the occurrence of α_j in phase t on process $p[l]$ must follow the occurrence of α_j in phase $t - D$ on process $p[l + L]$[2]. Similarly, a positive cycle of length L and period D implies that there exists an action α_i such that the occurrence of α_i in phase t on process $p[l]$ must precede the occurrence of α_i in phase $t - D$ on process $p[l + L]$. Consequently, as shown in Figure 4.6, the cycle condition (5.32) is necessary for a program to have CRT.

A simple consequence of the cycle condition is the following:

Lemma 5.2 *If a canonical pseudo-cyclic infinite-slack program $\mathcal{P}$ satisfies (5.32), then*

$$\nexists\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{oi} : (\Delta(\alpha_e, \alpha_f) = 1) \vee\ (\Delta(\alpha_e, \alpha_f) = 0 \wedge f < e), \tag{5.33}$$

$$\text{or} \qquad \nexists\, e, f : (\alpha_e, \alpha_f) \in \mathrm{C}_{io} : (\Delta(\alpha_e, \alpha_f) = 0 \wedge e < f). \tag{5.34}$$

[2] Recall Figure 4.7a.

To prove this lemma, note that the violation of (5.33) induces the positive cycle $\langle (e, f - n\Delta(\alpha_e, \alpha_f)), (e, f - n\Delta(\alpha_e, \alpha_f)) \rangle$ while the violation of (5.34) induces the negative cycle $\langle (e, f), (e, f) \rangle$ and both cycles are of length 1 and period 0.

If a canonical program $\mathcal{P}$ satisfies (5.33), then its dual is also canonical and satisfies (5.34). For programs that are deadlock-free and satisfies (5.34), we can modify Algorithm 2 to look for cycles[3]. For the zero-slack case, if no positive and negative cycles of the same length and period are found by the algorithm, then we can construct a CSSF for $\mathcal{P}$ and show that it has CRT. Unfortunately, we are unable to do the same for infinite-slack programs and therefore we can only conjecture that (5.30), (5.31), and (5.32) are also sufficient conditions for an infinite-slack program to have CRT. Further research is needed to prove this conjecture conclusively and thereby obtain a complete characterization of pseudo-cyclic infinite-slack programs with CRT.

[3]Basically, only edges in $C^{\star}_{io}$ ($C^{\star}_{oi}$) are considered in the construction of a negative (positive) path.

Chapter 6

Conclusion

In this paper we have presented the necessary and sufficient conditions for a linear array of processes, all implementing an identical program, to be deadlock-free and to have constant response times under the zero-slack communication model. All but one of these conditions is straightforward; also, an algorithm has been developed to check the remaining condition and, if possible, to produce a CSSF.

For infinite-slack communications, we have complete characterization of cyclic programs and can determine which psuedo-cyclic programs are deadlock-free. Furthermore, we have established several necessary conditions for a psuedo-cyclic program to have CRT and conjectured that they are also sufficient.

In the future, these results may be generalized to characterize the communication behavior of processes arranged in a topology other than a linear array. Therefore, we believe that the concepts in this paper constitute a useful framework for investigating the communication behavior of sets of communicating sequential processes.

Appendix A

Algorithms

A.1 Algorithm 1

Input: Cyclic program $\mathcal{P}$ satisfying (3.3) and (3.8).
Output: Integer M and set of integers $\{\pi(i) \mid 0 \leq i < n\}$ satisfying (3.9) and (3.10).

```
begin
    M := 0;    π(0) := 0;    k := 0;
    while (k < n − 1) do
        k := k + 1;
        π(k) := π(k − 1) + 1;
        if (k ∈ E(C)) then
            k' := fcorr(k, C);
            d := π(k) − π(k');
            if (d ≤ M) then
                π(k) := π(k') + M
            else
                δ := d − M;
                apply Procedure 1A to update M and {π(i) | 0 ≤ i < n}
            endif
        endif
    enddo
end
```

PROCEDURE 1A:

```
begin
    x := k − 1;
    while ({e | e ∈ E(C) ∧ e ≤ x} ≠ ∅) do
        ê := max{e | e ∈ E(C) ∧ e ≤ x};
        f̂ := fcorr(ê, C);
        ∀ i : f̂ < i ≤ k : π(i) := π(i) + δ;
        x := f̂
    enddo;
    M := M + δ
end
```

A.2 Algorithm 2

Input: A pseudo-cyclic canonical program $\mathcal{P}$ satisfying (4.18), (4.22), (4.23), and (4.24).
Output: The variable *status* which is set to cycles_found if $\mathcal{P}$ does not satisfy (4.20) and is set to no_cycles_found otherwise.

```
begin
  k := h;
  while (k < n − 1) do
      k := k + 1;
      if (k ∈ E(C°)) then
          v := h − n;    u := k;    w := k;
          B := {(e, f) | (e, f) ∈ C° ∧ e ≤ k};
          apply Procedure 2A on B to update status;
          if (status = cycles_found) then
              exit
          endif;
          v := −ψ(k);    u := ψ(h − n);    w := ψ(k − n − 1);
          B := {(ψ(f), ψ(e)) | (e, f) ∈ C° ∧ e ≤ k};
          apply Procedure 2A on B to update status;
          if (status = cycles_found) then
              exit
          endif
      endif
  enddo;
  status:= no_cycles_found
end
```

PROCEDURE 2A:

```
begin
  B* := {(e, f) | ∃ e, f, j : (ê, f̂) ∈ B ∧ j ∈ Z : e = ê + nj ∧ f = f̂ + nj};
  ex[0] := min{i | i ≥ w ∧ i ∈ E(B)};
  ey[0] := max{i | i < w ∧ i ∈ E(B)};
  κ := 0;
  while (true) do
      fx[κ] := fcorr(ex[κ]);
      fy[κ] := fcorr(ey[κ]);
      if (fx[κ] < ey[κ]) then
          ex[κ + 1] := min{i | i ≥ fx[κ] ∧ i ∈ E(B*)};
          ey[κ + 1] := max{i | i ≤ fy[κ] ∧ i ∈ E(B*)};
          κ := κ + 1;
          if (∃ 0 ≤ i < κ : ε(ex[i]) = ε(ex[κ]) ∧ ε(ey[i]) = ε(ey[κ])) then
              status:= cycles_found;
              exit
          endif
      else
          status:= no_cycles_found;
```

```
            exit
        endif
    enddo
end
```

A.3 Algorithm 3

Input: A pseudo-cycle canonical program $\mathcal{P}$ satisfying (4.18), (4.22), (4.23), and (4.24).
Output: The variable *status* is set to cycles_foundif $\mathcal{P}$ does not satisfy the cycle condition. Else, the algorithm returns integers T, M, and a set of integers $\{\eta(i) \mid h - n \leq i < n\}$ for which (4.30), (4.31), and (4.32) hold.

```
begin
    B := {(e + n − h, f + n − h) | (e, f) ∈ C° ∧ e ≤ h};
    apply Algorithm 1 on B to get M and {π(i) | 0 ≤ i < n + 1};
    ∀ i : h − n ≤ i ≤ h : η(i) = π(i + n − h);
    T := η(h) − η(h − n);    k := h;    status := no_cycles_found;
    while ((k < n − 1) ∧ (status = no_cycles_found)) do
        k := k + 1;
        η(k) = η(k − n) + T;
        if (k ∈ E(C°)) then
            k' := fcorr(k);
            d := η(k) − η(k');
            if (d > M) then
                δ := d − M;
                v := h − n;    u := k;    w := k;
                B := {(e, f) | (e, f) ∈ C° ∧ e ≤ k};
                ∀ i : h − n ≤ i ≤ k : π(i) := η(i);
                apply Procedure 3A on B to update status, T, M, and {π(i) | v ≤ i ≤ u};
                if (status = no_cycles_found) then
                    ∀ i : h − n ≤ i ≤ k : η(i) := π(i)
                endif
            elseif (d < M) then
                δ := M − d;
                η(k) := η(k) + δ;
                ∀ i : k − n ≤ i ≤ k : η(i) := η(i) + δ;
                T := T + δ;
                v := ψ(k);    u := ψ(h − n);    w := ψ(k − n − 1);
                B := {(ψ(f), ψ(e)) | (e, f) ∈ C° ∧ e ≤ k};
                ∀ i : h − n ≤ i ≤ k : π(ψ(i)) := −η(i);
                apply Procedure 3A on B to update status, T, M, and {π(i) | v ≤ i ≤ u};
                if (status = no_cycles_found) then
                    ∀ i : h − n ≤ i ≤ k : η(i) := −π(ψ(i))
                endif
            endif
        endif
    enddo
```

```
  end
```

PROCEDURE 3A:

```
  begin
    apply Procedure 2A to get status, κ, and {ex[i], fx[i], ey[i], fy[i] | 0 ≤ i ≤ κ};
    if (status = cycles_found) then
        exit
    else
        apply Procedure 3B to get f̂, M1 and to update T and {π(i) | v ≤ i < u};
        apply Procedure 3C to get M2 and to update T and {π(i) | v ≤ i < u};
        M := M + M1 + M2 + δ
    endif
  end
```

PROCEDURE 3B:

```
  begin
    j := 0;    M1 := 0;
    fz[0] := v + ⌊fx[κ]/n⌋ × n;
    while (∄ i : 0 ≤ i ≤ κ : φ(fz[j]) = φ(fx[i])) do
        ez[j] := ecorr(fz[j]);
        ê := ε(ez[j]);
        ∀ i : ê ≤ i ≤ u : π(i) := π(i) + δ;
        if (ê > n + v) then
            ∀ i : ê − n ≤ i ≤ u : π(i) := π(i) + δ;
            M1 := M1 + δ
        endif;
        T := T + δ;
        fz[j + 1] := min{i | i ≥ ez[j] ∧ i ∈ F(B*)};
        j := j + 1
    enddo;
    f̂ := φ(fz[j])
  end
```

PROCEDURE 3C:

```
  begin
    m := 0;
    while (f̂ ≠ φ(fx[m])) do
        x := φ(fx[m]);
        ∀ x ≤ i ≤ u : π(i) := π(i) + δ;
        if (x ≤ u − n) then
            ∀ i : x + n ≤ i ≤ u : π(i) := π(i) + δ;
            M2 := M2 + δ
        endif;
        T := T + δ;
        m := m + 1
    enddo
  end
```

A.4 Algorithm 4

Input: A pseudo-cycle canonical program $\mathcal{P}$ satisfying (4.18), (4.19), and (4.20).
Output: Integers T, M, and a set of integers $\{\eta'(i) \mid 0 \leq i < n' + 2n\}$ satisfying (4.36), (4.37), and (4.38).

```
begin
   apply Algorithm 3 to get M, T, and {η(i) | h − n ≤ i < n};
   ∀ i : 0 ≤ i < n : η'(i + n') := η(i);
   ∀ i : 0 ≤ i < n : η'(i + n' + n) := η(i) + T;
   k := n';
   while (k > 0) do
         k := k − 1;
         η'(k) = η'(k + 1) − 1;
         if (k ∈ F(C°)) then
            k' := ecorr(k);
            d := η'(k') − η'(k);
            if (d ≤ M) then
               η'(k) = η'(k) − M + d
            else
               f̂ := min{i | i ∈ F(C̄) ∧ i > k} ∪ {n'};
               m := f̂ − k;
               ∀ i : f̂ ≤ i < n' + 2n : η'(i) := η'(i) × m;
               ∀ i : k < i < f̂ : η'(i) := η'(f̂) − f̂ + i;
               η'(k) := η'(k') − M × m;
               M := M × m;
               T := T × m
            endif
         endif
   enddo
end
```

A.5 Algorithm 5

Input: Cyclic infinite-slack program $\mathcal{P}$ satisfying (5.6) and integer N.
Output: Set of integers $\{\tau[l](i)\}$ satisfying (5.7), (5.8), and (5.9).

```
begin
   N' := 1;   ∀ i : 0 ≤ i < n : τ[0](i) := i;
   while (N' < N) do
         N' := N' + 1;
         apply Procedure 5A to update {τ[l](i)}
   enddo
end
```

PROCEDURE 5A:

```
begin
```

$\forall\, l, i : 0 \le l < N' - 1 \wedge\ 0 \le i < n : \tau[l](i) = \tau[l](i) \times\ n;$
$k := -1;$
while $(\{e \mid \exists\, f :: (\alpha_e, \alpha_f) \in \mathrm{C}_{io}\ \wedge\ f > k\} \neq \emptyset)$ **do**
$\hat{e} := \min\{e \mid \exists\, f :: (\alpha_e, \alpha_f) \in \mathrm{C}_{io}\ \wedge\ f > k\};$
$\hat{f} := \mathbf{fcorr}(\hat{e}, \mathrm{C}_{io});$
$\forall\, i : k < i \le \hat{f} : \tau[N' - 1](i) := \tau[N' - 2](\hat{e}) - \hat{f} + i;$
$k := \hat{f}$
enddo;
$\forall\, i : k < i < n : \tau[N' - 1](i) := \tau[N' - 2](n - 1) + i - k - 1$
end

A.6 Algorithm 6

Input: Cyclic infinite-slack program $\mathcal{P}$ satisfying (5.6), (5.12), and (5.16).
Output: Integer M and set of integers $\{\pi(i) \mid 0 \le i < n\}$ satisfying (5.13), (5.14), and (5.15).

begin
$\mathrm{B}_{oi} := \{(\alpha_e, \alpha_f) \mid (\alpha_e, \alpha_f) \in \mathrm{C}_{oi}\ \wedge\ f < e\};$
$M := 0; \quad \pi(0) := 0; \quad k := 0;$
while $(k < n - 1)$ **do**
$k := k + 1;$
$\pi(k) := \pi(k - 1) + 1;$
if $(k \in \mathbf{E}(\mathrm{C}_{oi} \cup \mathrm{C}_{io}))$ **then**
$k' := \mathbf{fcorr}(k, \mathrm{C}_{oi} \cup \mathrm{C}_{io});$
$d := \pi(k) - \pi(k');$
if $((\alpha_k, \alpha_{k'}) \in \mathrm{C}_{io}\ \wedge\ d < M)$ **then**
$\pi(k) := \pi(k') + M$
endif;
if $((\alpha_k, \alpha_{k'}) \in \mathrm{B}_{oi}\ \wedge\ d > M)$ **then**
$\delta := d - M;$
apply Procedure 6A to update M and $\{\pi(i) \mid 0 \le i < n\}$
endif
endif
enddo
end

PROCEDURE 6A:
begin
$x := k - 1;$
while $(\{f \mid (\alpha_e, \alpha_f) \in \mathrm{C}_{io}\ \wedge\ e \le x\} \neq \emptyset)$ **do**
$\hat{f} := \max\{f \mid (\alpha_e, \alpha_f) \in \mathrm{C}_{io}\ \wedge\ e \le x\};$
$\hat{e} := \mathbf{ecorr}(\hat{f}, \mathrm{C}_{io});$
$\forall\, \hat{f} < i \le k : \pi(i) := \pi(i) + \delta;$
$x := \hat{f}$
enddo;

$M := M + \delta$
end

A.7 Algorithm 7

Input: Psuedo-cyclic infinite-slack canonical program $\mathcal{P}$ with two-phase expansion $\bar{\mathcal{P}}$ that satisfies (5.23).
Output: Integer T_N and set of integers $\{\tau[l](i)\}$ satisfying (5.24), (5.25), (5.26), (5.27), (5.28), and (5.29).

begin
$N' := 1; \quad T'_N := n;$
$\forall\, i : 0 \le i < n' + 2n : \tau[0](i) := i;$
while $(N' < N)$ do
$N' := N' + 1;$
apply Procedure 7A to update $\{\tau[l](i)\}$
enddo
end

PROCEDURE 7A:
begin
$\forall\, l, i : 0 \le l < N' - 1 \wedge\ 0 \le i < n : \tau[l](i) = \tau[l](i) \times\ (n' + n);$
$T_N := T_N \times\ (n' + n);$
$k := -1;$
while $(\{e \mid \exists\, f :: (\beta_e, \beta_f) \in \bar{\mathrm{C}}_{io}\ \wedge\ k < f < n' + n \wedge\ e < n' + n\} \neq \emptyset)$ do
$\hat{e} := \min\{e \mid \exists\, f :: (\beta_e, \beta_f) \in \bar{\mathrm{C}}_{io}\ \wedge\ k < f < n' + n \wedge\ e < n' + n\};$
$\hat{f} := \mathbf{fcorr}(\hat{e}, \bar{\mathrm{C}}_{io});$
$\forall\, i : k < i \le \hat{f} : \tau[N' - 1](i) := \tau[N' - 2](\hat{e}) - \hat{f} + i;$
$k := \hat{f}$
enddo;
$u := \max(\{e \mid \exists\, f :: (\beta_e, \beta_f) \in \bar{\mathrm{C}}_{oi}\ \wedge\ k < f < n' + n\} \cup\ \{n' + n - 1\});$
$\forall\, i : k < i < n' + n : \tau[N' - 1](i) := \tau[N' - 2](u) + i - k;$
$\forall\, i : n' + n \le i < n' + 2n : \tau[N' - 1](i) := \tau[N' - 1](i - n) + T_N$
end

Bibliography

[1] C. A. R. Hoare, "Communicating Sequential Processes," *Comm. ACM* 21, 8, pp 666-677 (1978).

[2] H. T. Kung and C. E. Leiserson, "Algorithms for VLSI Processor Arrays", Section 8.3 in Mead & Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.

[3] S. Y. Kung, *VLSI Array Processors*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[4] A. J. Martin, "An Axiomatic Definition of Synchronization Primitives," *Acta Informatica*, 16, 1981, pp 219-235.

[5] A. J. Martin, "The Probe: An Addition to Communication Primitives," *Information Processing Letters*, 20, 1985, pp 125-130.

[6] M. Rem, "Trace Theory and Systolic Computations," Caltech Computer Science Technical Report 5239:TR:87, 1987.